

# به نام دادار دادآفرین

مینی پروژه دوم درس مبانی سیستم‌های هوشمند

استاد درس: دکتر مهدی علیاری

گردآورنده: امیر جهانگرد تکالو

شماره دانشجویی: ۹۹۲۵۲۹۳



۱۳۰۷

دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده مهندسی برق

## فهرست

۳	سوال ۱
۳	قسمت ۱-۱
۷	قسمت ۲-۱
۹	قسمت ۳-۱
۱۴	سوال ۲
۱۴	قسمت ۱-۲
۱۹	قسمت ۲-۲
۲۵	سوال ۳
۲۵	قسمت ۱-۳
۳۲	قسمت ۲-۳
۴۲	قسمت ۳-۳
۵۴	سوال ۴
۵۵	قسمت ۱-۴
۵۸	قسمت ۲-۴
۶۲	قسمت ۳-۴
۶۶	قسمت ۴-۴
۶۷	قسمت ۵-۴
۷۲	قسمت ۶-۴
۷۶	قسمت ۷-۴
۷۸	قسمت ۸-۴
۸۶	سوال ۵
۸۶	قسمت ۱-۵
۹۴	قسمت ۲-۵

# سوال ۱

مجموعه داده مربوط به این سوال را از طریق این پیوند دانلود کنید و در مراحل بعدی از آن استفاده کنید. ستون اول و دوم فایل CSV مربوط به این مجموعه داده، مربوط به ویژگی‌ها و ستون سوم آن مربوط به کلاس هر داده است.

## قسمت ۱-۱

داده‌ها را با نسبت ۸۰ به ۲۰ درصد به دو قسمت آموزش و آزمون تقسیم کنید. سپس با استفاده از قاعدة پرسپترون، یک نورون روی داده‌های مجموعه آموزشی، آموزش دهید (آستانه را دلخواه در نظر بگیرید).

ابتدا کتابخانه‌های مذکور را فراخوانی می‌کنیم که عبارتند از:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
```

از کتابخانه برای تقسیم داده‌ها به آموزش و تست استفاده می‌کنیم.

از کتابخانه الگوریتم پرسپترون را فراخوانی می‌کنیم.

```
#https://drive.google.com/file/d/1OKMqLEUFQvMnjDJ1tfr4oXfXk2uIKk2O/view?usp=sharing
!gdown 1OKMqLEUFQvMnjDJ1tfr4oXfXk2uIKk2O
```

در این قسمت از کد ما فایل اکسل پایگاه داده‌مان را به پایتون می‌فرستیم.

```
# part 1
datasets = pd.read_csv('/content/Perceptron.csv')
datasets
```

در این بخش از کد، یک فایل csv به نام 'Perceptron.csv' از مسیر '/content/' خوانده شده و در یک متغیر به نام 'datasets' ذخیره شده است. سپس این دیتاست را فراخوانی می‌کنیم و خروجی را مشاهده می‌کنیم:

	x1	x2	y
0	1.028503	0.973218	-1.0
1	0.252505	0.955872	-1.0
2	1.508085	0.672058	-1.0
3	1.940002	1.721370	-1.0
4	-1.048819	-0.844999	1.0
...	...	...	...
395	0.574634	0.782211	-1.0
396	-1.413307	-0.673049	1.0
397	-0.465114	-1.290830	1.0
398	1.522055	0.948007	-1.0
399	0.834118	0.926710	-1.0

400 rows × 3 columns

می‌بینیم که فایل اکسل ما شامل ۴۰۰ سطر و ۳ ستون می‌باشد که ۲ ستون اول همان X ما و ستون سوم همان y ما می‌باشد. این توضیحات را در پایتون نیز اعمال می‌کنیم:

```
X = datasets[['x1', 'x2']]
y = datasets['y']
print(X, y)
```

در این بخش از کد، داده‌های ورودی X و خروجی y از مجموعه داده‌ها استخراج شده است. داده‌های ورودی شامل ستون‌های x1 و x2 از مجموعه داده‌ها بوده و داده‌های خروجی نیز متعلق به ستون y می‌باشد. سپس با پرینت گرفتن خروجی نتایج را مشاهده می‌کنیم:

```

      x1          x2
0    1.028503  0.973218
1    0.252505  0.955872
2    1.508085  0.672058
3    1.940002  1.721370
4   -1.048819 -0.844999
..
395  0.574634  0.782211
396 -1.413307 -0.673049
397 -0.465114 -1.290830
398  1.522055  0.948007
399  0.834118  0.926710

[400 rows x 2 columns] 0      -1.0
1      -1.0
2      -1.0
3      -1.0
4       1.0
...
395     -1.0
396      1.0
397      1.0
398     -1.0
399     -1.0
Name: y, Length: 400, dtype: float64

```

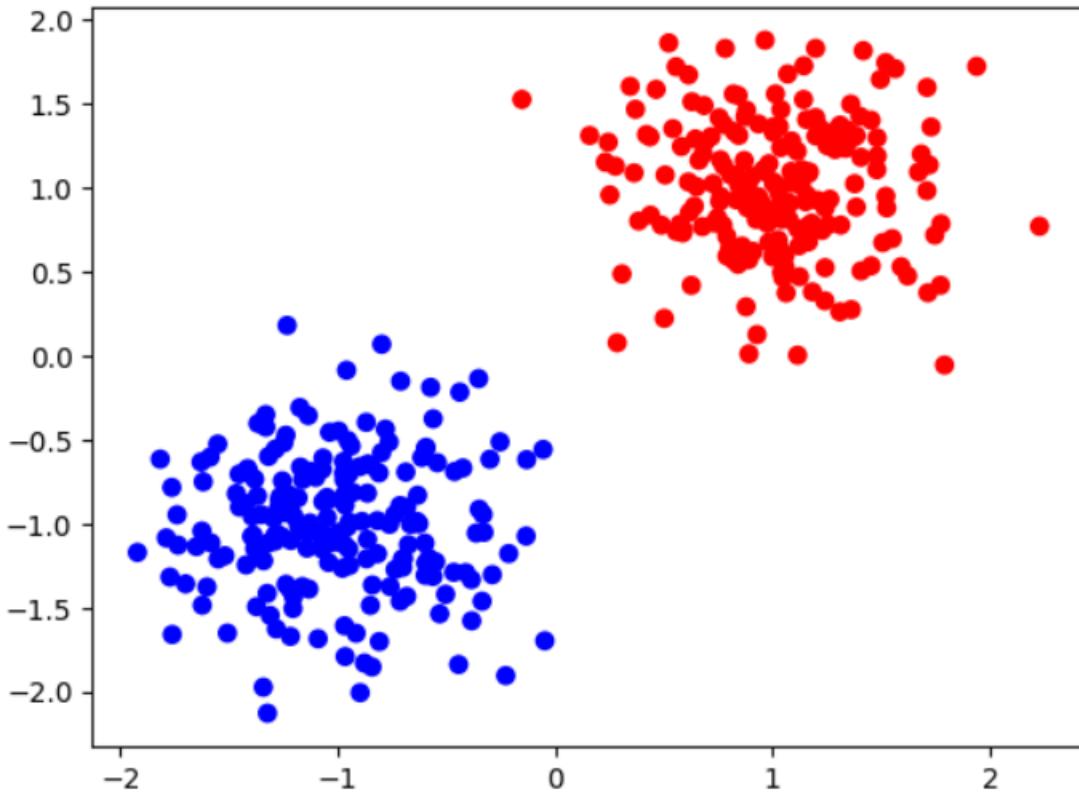
در ادامه می خواهیم توزیع داده ها را روی نمودار مشاهده کنیم:

```

colors = ['blue' if label == 1 else 'red' for label in y]
a = datasets['x1']
b = datasets['x2']
plt.scatter(a, b, c=colors)
plt.show

```

در سطر اول برای تفکیک خروجی هایی که مقدارشان ۱ و -۱ است، از رنگ های آبی و قرمز استفاده کردیم، به طوری که با یک خط دستور شرطی if تعیین کردیم که اگر مقدار نهایی ما ۱ باشد، رنگ مورد نظر ما آبی می باشد و در غیر این صورت داده ما به رنگ قرمز است. حال خروجی را مشاهده می کنیم:



حال می‌توانیم داده‌ها را تقسیم‌بندی کنیم:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 93)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

در این بخش از کد، داده‌های ورودی و خروجی به صورت تصادفی به دو مجموعه آموزش و آزمون تقسیم شده‌اند. مجموعه آزمون ۲۰ درصد از کل داده‌ها را شامل می‌شود و مجموعه آموزش ۸۰ درصد باقیمانده را شامل می‌شود. این تقسیم داده‌ها با استفاده از تابع `train_test_split` انجام شده است. همچنین، با فراخوانی `shape` برای هر چهار متغیر `X_train`, `X_test`, `y_train` و `y_test`، ابعاد هر کدام از آن‌ها نمایش داده شده است:

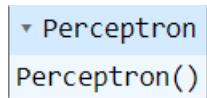
```
((320, 2), (80, 2), (320,), (80,))
```

حال پس از اینکه داده‌ها را تقسیم کردیم، می‌توانیم یک نورون روی داده‌ها آموزش بدھیم:

```
perceptron_model = Perceptron()
# threshold = 0
perceptron_model.fit(X_train, y_train)
```

در این بخش از کد، یک مدل پرسپترون ایجاد شده و سپس با استفاده از داده‌های آموزش X\_train و y\_train مدل آموزش داده شده است. این کد مربوط به فرآیند آموزش مدل پرسپترون بر روی داده‌های آموزش است. باید توجه شود که در این قسمت از سوال آستانه صفر در نظر گرفته شده است.

نمایش خروجی:



## ۲-۱ قسمت

نتیجه را روی داده‌های مجموعه آزمون نشان دهید و دقت را به دست آورید. برای داده‌های تست دو خط موازی جداکننده به دست آمده از قاعده پرسپترون را نمایش دهید و داده‌های تفکیک شده دو کلاس را با رنگ مجزا در scatter plot مشخص کنید

```
# part 2
test_accuracy = perceptron_model.score(X_test, y_test)
test_accuracy
```

در این بخش از کد، دقت مدل پرسپترون بر روی داده‌های آزمون محاسبه شده و در متغیر test\_accuracy ذخیره شده است. این دقت نشان دهنده عملکرد مدل بر روی داده‌های جدید است.

با فراخوانی test\_accuracy داریم:

1.0

پس مشاهده می‌کنیم دقت کار ما ۱۰۰ درصد است و هیچ خطای نداریم، پس نورون ساخته شده ما مناسب است.

حال به تفکیک دو کلاس می‌پردازیم:

```
X_test_array = X_test.values
x_min, x_max = X_test_array[:, 0].min() - 1, X_test_array[:, 0].max() + 1
y_min, y_max = X_test_array[:, 1].min() - 1, X_test_array[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))
Z = perceptron_model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
```

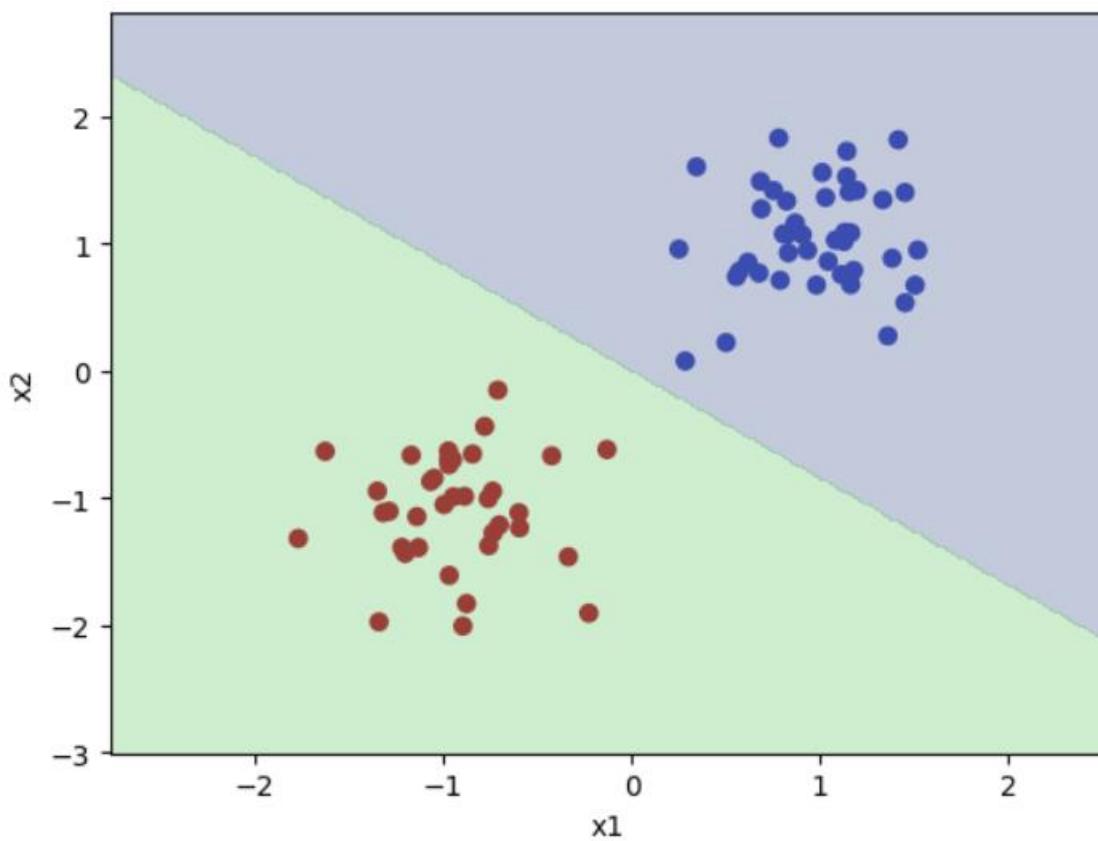
```

levels = [-1,0,1]
plt.scatter(X_test_array[:, 0], X_test_array[:, 1], c=y_test,
cmap=plt.cm.coolwarm)
plt.contourf(xx, yy, Z, levels=levels, alpha=0.3)
plt.xlabel('x1')
plt.ylabel('x2')
plt.show()

```

در این بخش از کد، با استفاده از داده‌های آزمون، محدوده‌ای برای نمایش محور  $x$  و  $y$  ایجاد شده است. سپس با استفاده از تابع `meshgrid`، یک مربع شبکه‌ای با فاصله‌ی  $0.2 \times 0.2$  بین نقاط ایجاد شده است. سپس با استفاده از مدل پرسپترون، برای هر نقطه در این شبکه، خروجی مدل محاسبه شده و در متغیر  $Z$  ذخیره شده است. سپس با استفاده از تابع `contourf`، ناحیه‌های مختلف برای نمایش داده‌ها ایجاد شده است. از عبارت `levels = [-1,0,1]` برای رسم ۲ خط موازی استفاده می‌شود. در ادامه با استفاده از تابع `scatter`، داده‌های آزمون روی نمودار نمایش داده شده‌اند. در نهایت محور  $x$  و  $y$  به ترتیب با  $x1$  و  $x2$  برچسب‌گذاری شده‌اند.

حال می‌توان در خروجی که داده‌های آزمون را تفکیک کردیم مشاهده کنیم:



در این سوال از ما خواسته شد که متغیرها را با ۲ خط موازی تفکیک کنیم که با استفاده از در کد این کار را اعمال کردیم، اما به علت نزدیکی خطها و همچنین همنگ بودن خطها با صفحه به خوبی قابل تشخیص نیست.

### قسمت ۱-۳

قسمت‌های ۱ و ۲ را با آستانه دیگر انجام داده و نتایج را با حالت قبل مقایسه کنید. تحلیل کنید که آستانه در پرسپترون چه تأثیری روی نتایج طبقه‌بندی دارد. ضمن پیاده‌سازی تحلیل کنید که حذف بایاس چه تأثیری بر نتایج خواهد گذاشت.

آستانه را یک عدد جدید انتخاب می‌کنیم، مثلاً ۵٪.

```
# part 3
perceptron_model_new = Perceptron(random_state=42, tol=0.001,
max_iter=1000, eta0=0.1, verbose=0, n_jobs=-1)
perceptron_model_new.fit(X_train, y_train)
```

برای آموزش مدل پرسپترون با استفاده از داده‌های آموزش، یک نمونه جدید از مدل که آستانه آن ۵٪ است، با استفاده از کتابخانه Scikit-learn ایجاد شد. این مدل با استفاده از پارامترهای زیر تنظیم شد:

random\_state=42, tol=0.001 max\_iter=1000, eta0=0.1, verbose=0, n\_jobs=-1

سپس مدل بر روی داده‌های آموزش X\_train و y\_train آموزش داده شد. این مدل جدید به صورت perceptron\_model\_new ایجاد شد.

در ادامه همه کارهایی که در قسمت قبلی انجام دادیم را تکرار می‌کنیم:

```
test_accuracy_1 = perceptron_model_new.score(X_test, y_test)
test_accuracy_1
```

مشاهده خروجی:

1.0

```
Z = perceptron_model_new.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

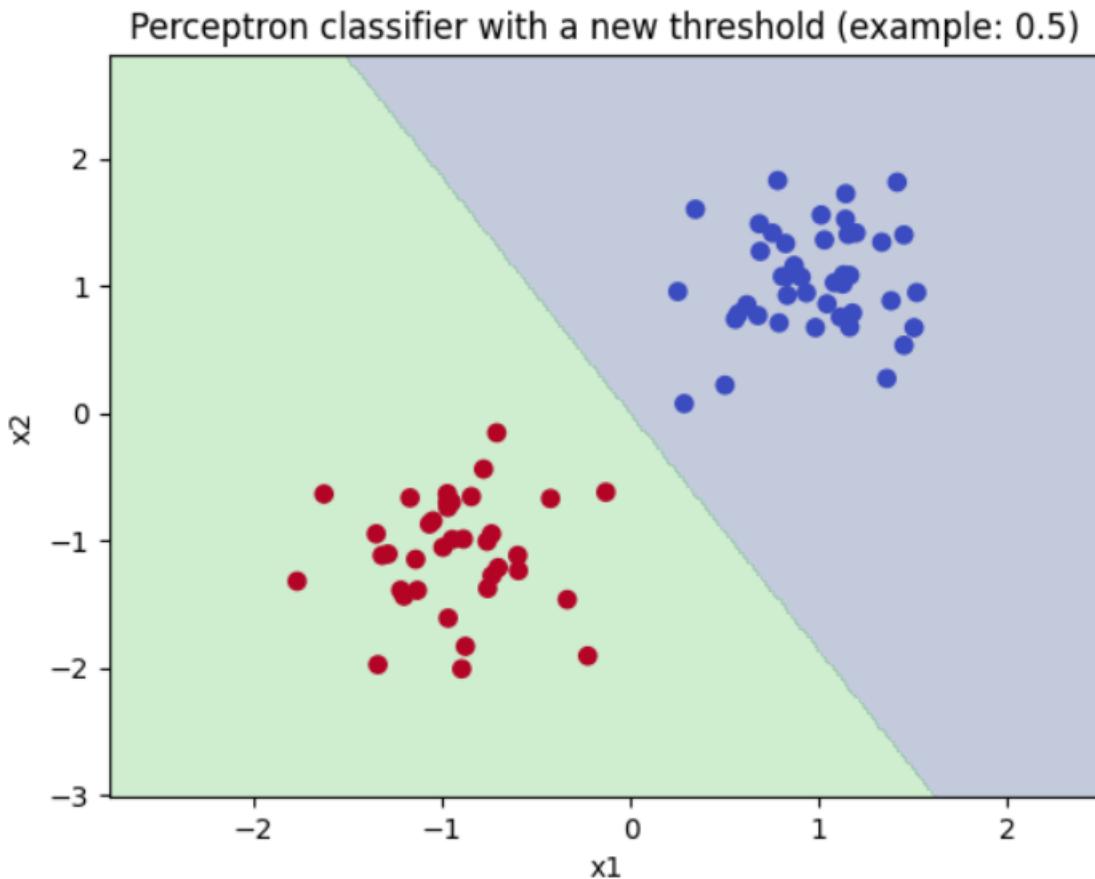
plt.contourf(xx, yy, Z, levels=levels, alpha=0.3)
plt.scatter(X_test_array[:, 0], X_test_array[:, 1], c=y_test,
cmap=plt.cm.coolwarm)
```

```

plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Perceptron classifier with a new threshold (example: 0.5)')
plt.show()

```

مشاهده خروجی:



مقایسه خروجی و توضیح تأثیر انتخاب آستانه در پرسپترون:

انتخاب آستانه (threshold) در الگوریتم پرسپترون بر طبقه‌بندی نتایج تأثیر مهمی دارد. آستانه به عنوان یک مقدار تصمیم‌گیری استفاده می‌شود که مشخص می‌کند که خروجی مدل به عنوان کلاس مثبت یا منفی تشخیص داده شود.

تأثیر انتخاب آستانه بر نتایج طبقه‌بندی به شرح زیر است:

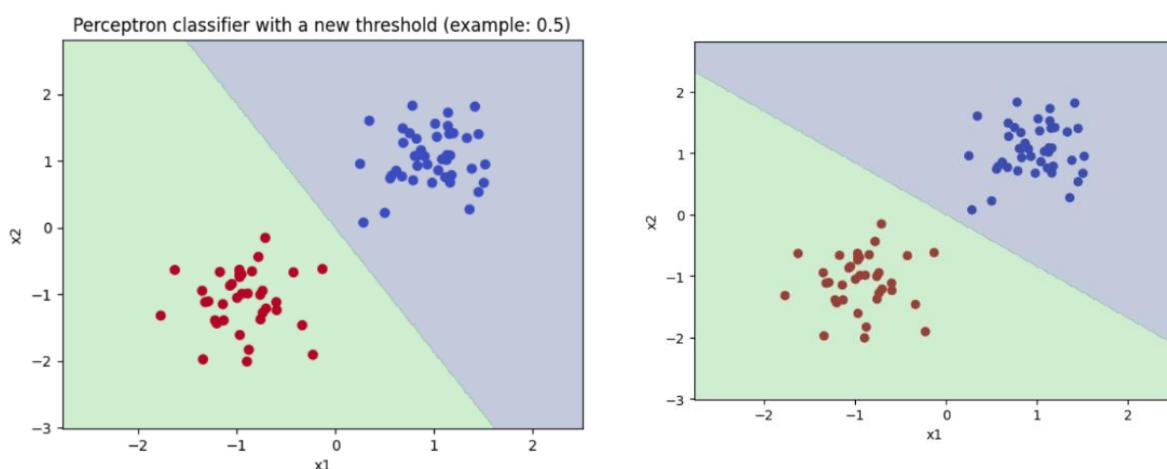
۱. تعیین دقต: انتخاب آستانه می‌تواند تأثیر مستقیمی بر دقت طبقه‌بندی داشته باشد. زیرا انتخاب آستانه منجر به تعیین تعداد نمونه‌هایی می‌شود که به درستی طبقه‌بندی شده‌اند یا نه.

۲. تعیین تعادل بین حساسیت و ویژگی خاصیت: انتخاب آستانه می‌تواند تعادل بین حساسیت (توانایی تشخیص نمونه‌های مثبت) و ویژگی خاصیت (توانایی تشخیص نمونه‌های منفی) را تعیین کند. این موضوع می‌تواند در موقعي که ترجیح داده می‌شود یک نوع خطرا بر نوع دیگر ترجیح دهیم، بسیار مهم باشد.

۳. تأثیر بر توانایی تشخیص: انتخاب آستانه می‌تواند تأثیر مستقیمی بر توانایی مدل در تشخیص نمونه‌های مثبت یا منفی داشته باشد. آستانه‌های مختلف می‌توانند تعداد نمونه‌هایی که به درستی تشخیص داده می‌شوند را تغییر دهند.

بنابران، انتخاب آستانه یک جزئی مهم از فرآیند طبقه‌بندی با پرسپترون است و می‌تواند تأثیر بزرگی بر نتایج نهایی داشته باشد. انتخاب آستانه بهترین کاری است که ممکن است نیاز به تنظیم و تجربه داشته باشد.

در این سوال می‌بینیم که تفکیک کننده ما به خوبی کار می‌کند و در هر دو آستانه مدنظر ما، نتیجه به درستی نمایش داده شده و هیچ داده پرتی نداریم:



تنها مورد متفاوت، خطی است که داده‌های ما از هم تفکیک می‌کند، که مشاهده می‌شود با آستانه  $5^{\circ}$  رشیب خط بهتر شده و به عبارتی در این حالت صفحه ما به ۲ بخش مساوی تری تقسیم می‌شود.

در مرحله بعد برای حذف بایاس، ابتدا مدل جدیدی می سازیم:

```
perceptron_model_without_bias = Perceptron(fit_intercept=False,  
random_state=42, tol=0.001, max_iter=1000, eta0=0.1, verbose=0, n_jobs=-1)  
perceptron_model_without_bias.fit(X_train, y_train)
```

مشاهده خروجی:

```
▼  
Perceptron  
Perceptron(eta0=0.1, fit_intercept=False, n_jobs=-1, random_state=42)
```

```
test_accuracy_2 = perceptron_model_without_bias.score(X_test, y_test)  
test_accuracy_2
```

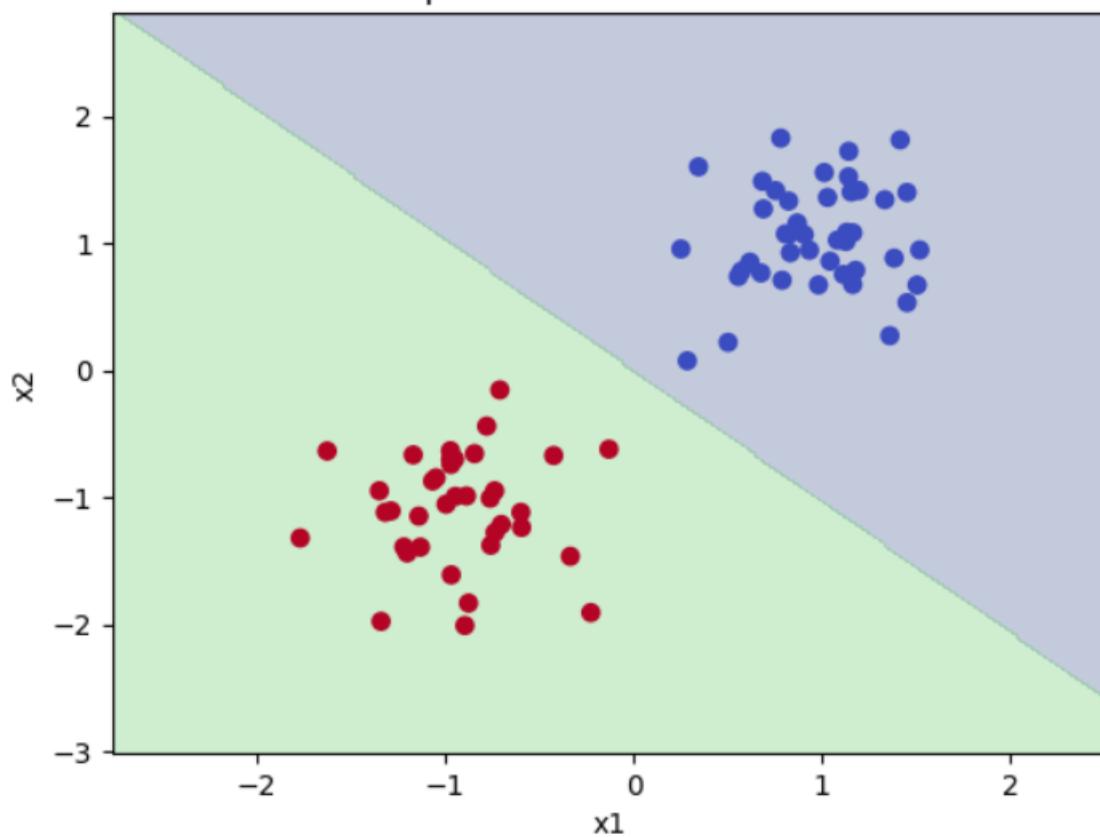
مشاهده خروجی:

```
1.0
```

```
Z = perceptron_model_without_bias.predict(np.c_[xx.ravel(), yy.ravel()])  
Z = Z.reshape(xx.shape)  
  
plt.contourf(xx, yy, Z, levels=levels, alpha=0.3)  
plt.scatter(X_test_array[:, 0], X_test_array[:, 1], c=y_test,  
cmap=plt.cm.coolwarm)  
plt.xlabel('x1')  
plt.ylabel('x2')  
plt.title('Perceptron classifier without bias')  
plt.show()
```

حال خروجی را در حالت حذف بایاس مشاهده می کنیم:

Perceptron classifier without bias



مشاهده می شود که در این حالت صفحه با دقت بسیار بالایی به نصف تقسیم شده است.

## سوال ۲

### قسمت ۱-۲

به کمک نورون McCulloch-Pitts توسعه یافته، یک ضرب کننده باینری بسازید که دو ورودی دویتی را گرفته و آن ها را ضرب کند برای این کار به دو ورودی دویتی (در واقع چهار نورون برای همه ورودی ها) نیاز داریم. همچنان چهار بیت خروجی (چهار نورون) مورد نیاز است. توجه شود که تمامی نورون های ورودی و خروجی باینری هستند (صفر و یک). ترتیب زمانی انجام عملیات در این سوال مهم نیست؛ بنابراین، نیازی به در نظر گرفتن تأخیر برای انجام عملیات نیست.

ضمن رسم جدول ورودی-خروجی، شبکه هر خروجی را به همراه توضیحات مختصراً رسم کنید (نیازی به کدنویسی در این قسمت نیست). دقت داشته باشید که شبکه ای که برای هر خروجی رسم می کنید تا حد ممکن دارای کمترین تعداد نورون و کمترین آستانه باشد (تعداد نورون کمتر دارای اهمیت بالاتری نسبت به آستانه کوچکتر است). همچنان توجه کنید که تمام شبکه برای یک خروجی دارای آستانه یکسان باشد.

برای این سوال به منظور درک بیشتر ابتدا ورودی ها را تک بیتی در نظر می گیریم و مثال را برای گیت XNOR حل می کنیم.

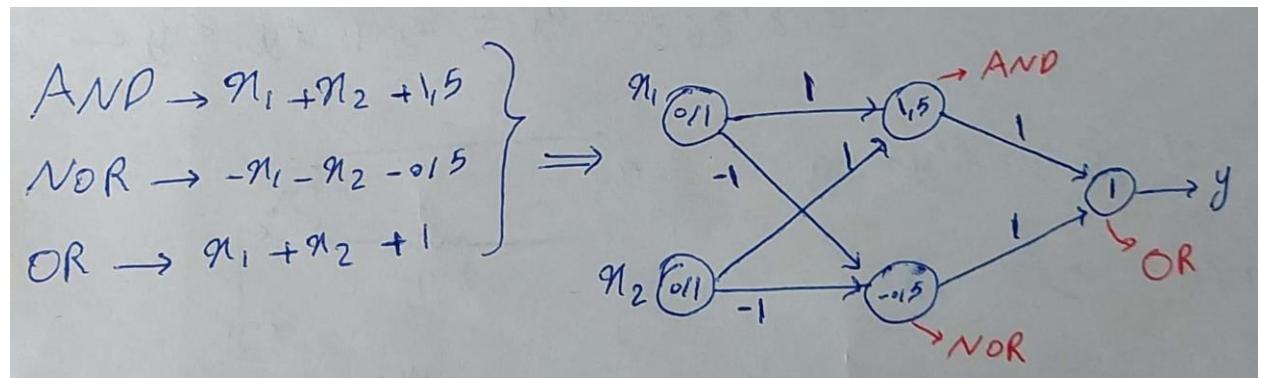
جدول ورودی-خروجی این گیت را بررسی می کنیم:

XNOR $\Rightarrow$		$n_1$	$n_2$	$y = \overline{n_1 \oplus n_2}$
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	1	1	1

حال تابع خروجی را بررسی می کنیم و هدفمان این است که بدانیم از کدام یک از گیت های منطقی قرار است استفاده کنیم:

$$y = \pi_1 \pi_2 + \bar{\pi}_1 \bar{\pi}_2 = \underbrace{\pi_1 \pi_2}_{AND} + \underbrace{\frac{\pi_1 + \pi_2}{\pi_1 + \pi_2}}_{NOR} \underbrace{+ \frac{\pi_1 + \pi_2}{\pi_1 + \pi_2}}_{OR}$$

حال وزن ها را انتخاب می کنیم و مدارنهایی را رسم می کنیم:



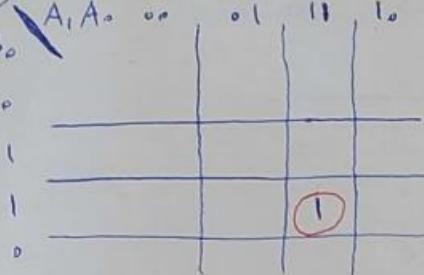
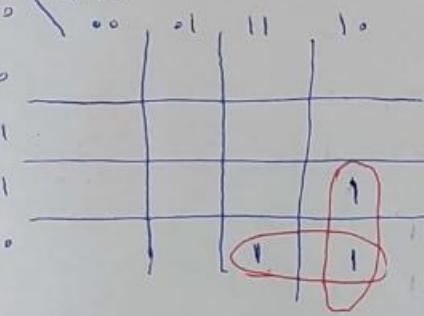
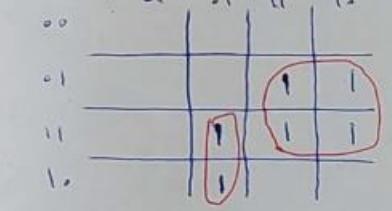
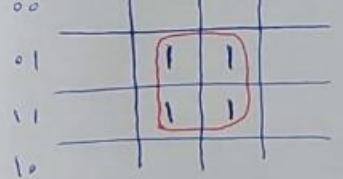
حال یک ضرب کننده را بررسی می کنیم که ۲ تا ورودی ۲ بیتی و ۴ تا خروجی دارد:

ابتدا جدول ورودی-خروجی این ضربکننده را رسم می کنیم:

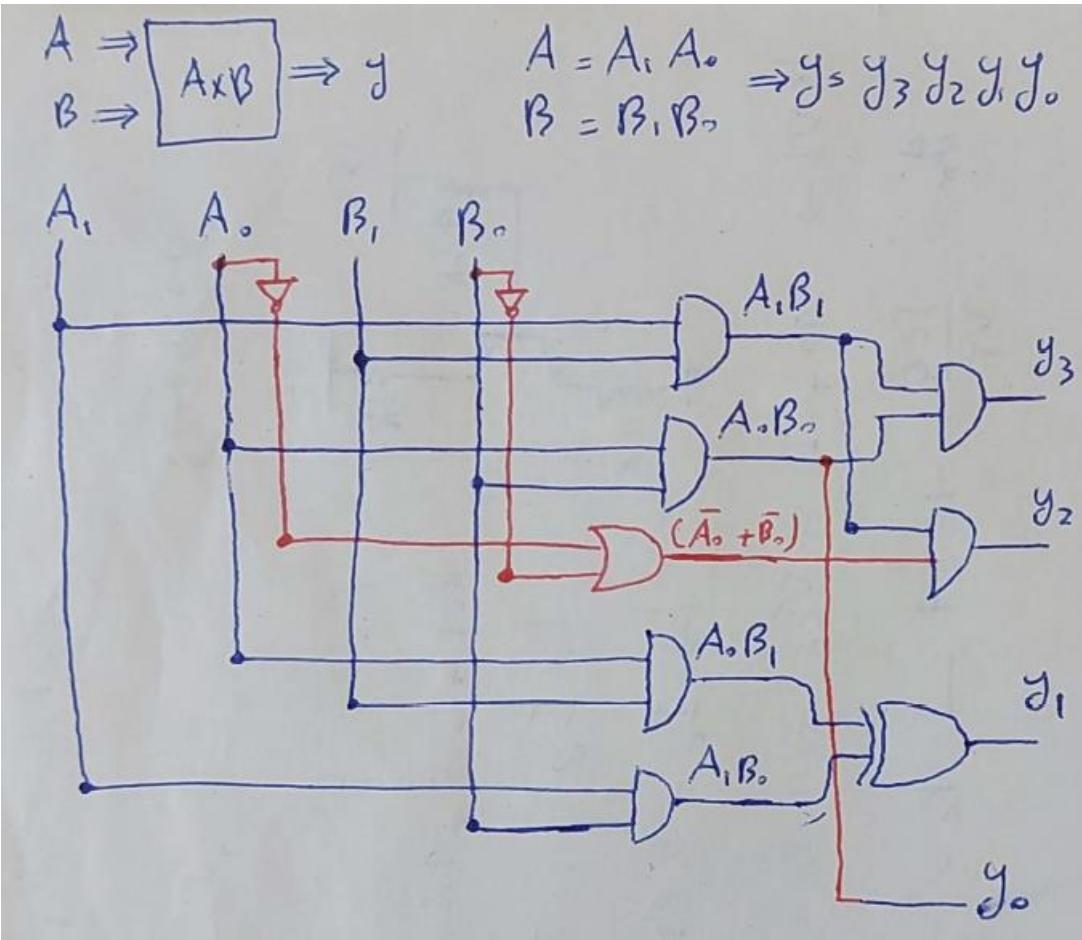
جدول حقیقت:

$y = A \times B$		B		y			
A	A.	B.	B.	y <sub>3</sub>	y <sub>2</sub>	y <sub>1</sub>	y <sub>0</sub>
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

مشاهده می شود که ۴ تا خروجی داریم. حال تابع هر کدام از خروجی ها را بررسی می کنیم:

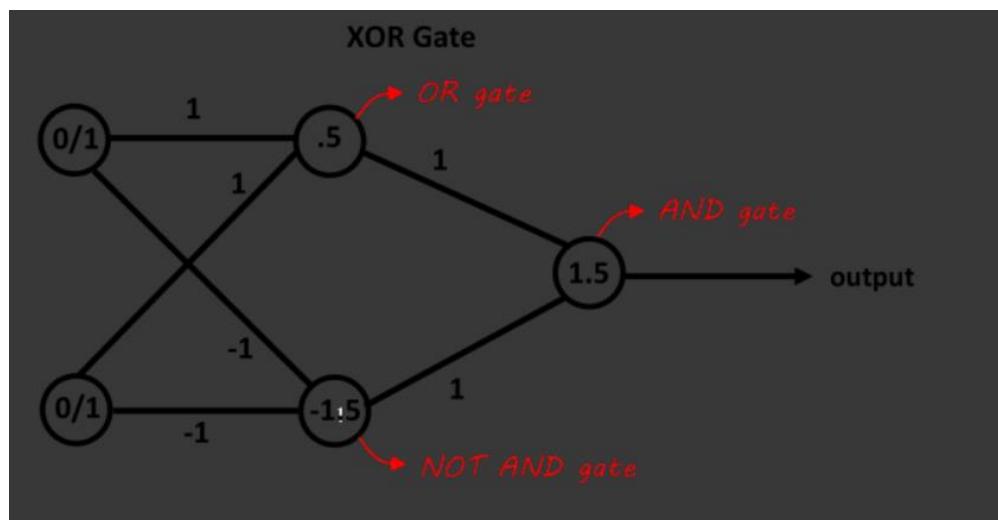
$y_3$	$A_1 A_0 \setminus B_1 B_0$		$\Rightarrow y_3 = A_1 A_0 B_1 B_0$
$y_2$	$B_1 B_0 \setminus A_1 A_0$		$\Rightarrow y_2 = A_1 \bar{A}_0 B_1 + A_1 B_0 \bar{B}_0$ $\hookrightarrow y_2 = A_1 B_1 (\bar{A}_0 + \bar{B}_0)$
$y_1$	$B_1 B_0 \setminus A_1 A_0$		$\Rightarrow y_1 = A_1 B_0 + \bar{A}_1 A_0 B_1$ $\hookrightarrow y_1 = A_0 B_1 \oplus A_1 B_0$
$y_0$	$B_1 B_0 \setminus A_1 A_0$		$\Rightarrow y_0 = A_0 B_0$

در نهایت با توجه به توابع مدار نهایی را بر اساس گیت‌های NOT، AND، OR و XOR رسم می‌کنیم:

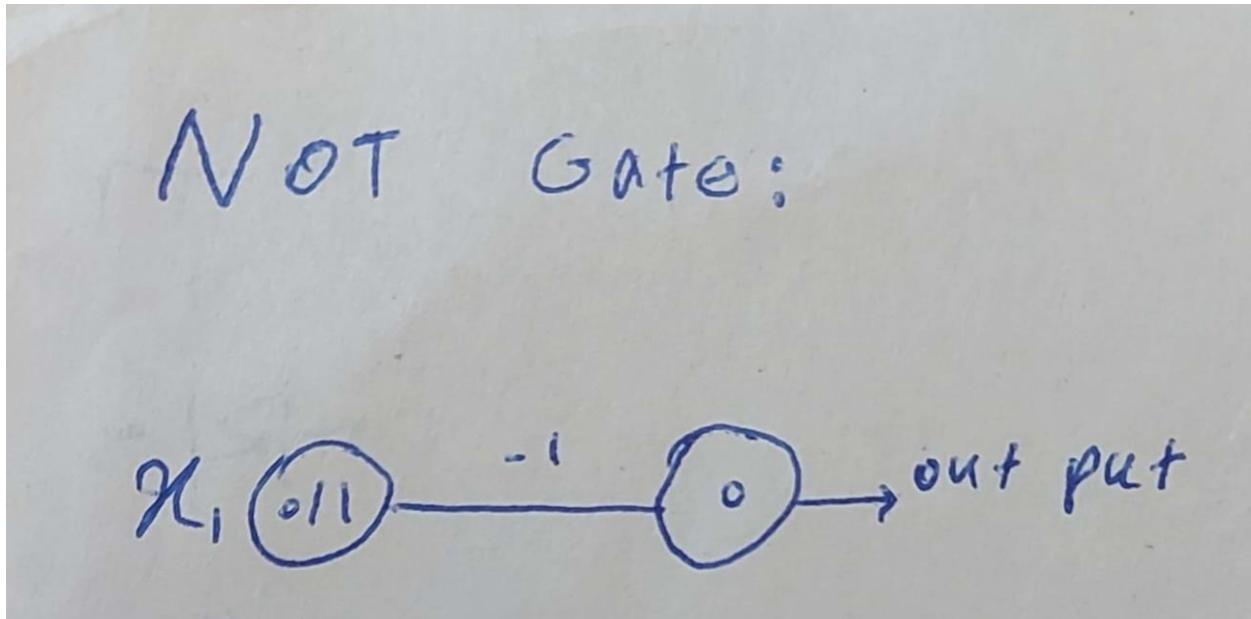


حال می‌توان مثل حالت قبل وزن‌ها را انتخاب کرد. در مثال قبل وزن‌های مناسب را برای گیت‌های AND و OR انتخاب کردیم، حال وزن‌های مناسب را برای گیت‌های NOT و XOR انتخاب می‌کنیم:

گیت XOR:



گیت NOT :



حال با داشتن وزن‌ها می‌توان به کدنویسی پرداخت.

## ۲-۳ قسمت

با استفاده از زبان پایتون شبکه‌های طراحی شده در قسمت ۱ را پیاده‌سازی کرده و تمامی حالات ممکن را بصورت مناسبی نمایش دهید.

ابتدا گیت XNOR را بررسی می‌کنیم:

```
import numpy as np
import itertools
```

این دو خط کد مربوط به وارد کردن کتابخانه NumPy و ماژول itertools در پایتون است.  
ماژول itertools شامل توابع مفیدی برای کار با داده‌های قابل تکرار (iterable) مانند لیست‌ها، تاپل‌ها و ... است.  
این ماژول شامل توابعی برای ترکیب‌ها، ترتیب‌ها، تکرارها و ... می‌باشد.

```
class McCulloch_Pitts_neuron():

    def __init__(self, weights, threshold):
        self.weights = weights
        self.threshold = threshold
```

```

def model(self , x):
    if self.weights @ x >= self.threshold:
        return 1
    else:
        return 0

```

این کد یک مدل ساده از یک نورون مصنوعی را پیاده‌سازی می‌کند که می‌تواند برای مسائل ساده‌تر یادگیری ماشین و شبکه‌های عصبی مصنوعی استفاده شود.

این کد یک کلاس به نام McCulloch\_Pitts\_neuron ایجاد می‌کند که این نورون را پیاده‌سازی می‌کند. این نورون دو ویژگی اصلی دارد: وزن‌ها (weights) و آستانه (threshold).

در متدهای `__init__`, وزن‌ها و آستانه به عنوان ورودی‌های کلاس گرفته می‌شوند و در متغیرهای `weights` و `threshold` ذخیره می‌شوند.

در متدهای `model`, ورودی `x` به عنوان ورودی نورون گرفته می‌شود و با استفاده از محاسبه‌ی ضرب داخلی بین وزن‌ها و ورودی (`self.weights @ x`), مقدار خروجی نورون محاسبه می‌شود. اگر این مقدار بیشتر یا مساوی آستانه باشد، خروجی ۱ و در غیر این صورت، خروجی ۰ خواهد بود.

```

def XNOR(input):
    neur1 = McCulloch_Pitts_neuron([1, 1], 1.5)
    neur2 = McCulloch_Pitts_neuron([-1, -1], -0.5)
    neur3 = McCulloch_Pitts_neuron([1, 1], 1)

    z1 = neur1.model(np.array([input[0], input[1]]))
    z2 = neur2.model(np.array([input[0], input[1]]))
    z3 = neur3.model(np.array([z1, z2]))

    return list([z1, z2, z3])

```

این کد یک تابع به نام XNOR با استفاده از ۳ نورون McCulloch\_Pitts پیاده‌سازی می‌کند.

در این تابع، سه نورون McCulloch\_Pitts با وزن‌های مشخص و آستانه‌های مشخص که در قسمت ۱ تعریف کردیم ایجاد می‌شوند. سپس با استفاده از ورودی‌های تابع (`input`), خروجی‌های هر یک از نورون‌ها محاسبه می‌شوند و در متغیرهای `z1`, `z2` و `z3` ذخیره می‌شوند.

در نهایت، خروجی تابع به صورت یک لیست از سه مقدار  $z_1$ ,  $z_2$  و  $z_3$  برگردانده می‌شود. این سه مقدار به ترتیب برابر با خروجی‌های نورون‌های اول، دوم و سوم در مدار XNOR هستند که دیدیم در مدار ما AND و NOR و XNOR را به دست آورد. نام‌گذاری شدند. با ترکیب این سه خروجی، می‌توان خروجی نهایی مدار XNOR را به دست آورد.

```
input = [1, 0]
X = list(itertools.product(input, input))

for i in X:
    res = XNOR(i)
    print("XNOR with input as", str(i[0]) + " ")+str(i[1]), "goes to
output ", str(res[2]))
```

این قطعه کد از تابع XNOR استفاده می‌کند تا خروجی XNOR را برای تمامی حالت‌های ورودی ممکن محاسبه کند.

در اینجا ابتدا تمامی حالت‌های ورودی ممکن برای ورودی  $[1, 0]$  با استفاده از تابع `itertools.product` ایجاد می‌شود. سپس برای هر یک از این حالت‌ها، خروجی تابع XNOR محاسبه شده و چاپ می‌شود.

حال می‌توان خروجی را مشاهده کرد:

```
XNOR with input as 1 1 goes to output 1
XNOR with input as 1 0 goes to output 0
XNOR with input as 0 1 goes to output 0
XNOR with input as 0 0 goes to output 1
```

مشاهده می‌شود که خروجی بدست آمده از پایتون با خروجی بدست آمده از محاسبات دستی ما یکسان است.

حال به بررسی ضرب‌کننده‌مان می‌پردازیم:

ابتدا برای ۴ خروجی که داریم، هر کدام را بر اساس وزن‌ها و آستانه‌هایی که تعریف کردیم، جداگانه تعریف می‌کنیم:

:Y3 تعریف

```
def Y3(input):
    neur1 = McCulloch_Pitts_neuron([1, 0, 1, 0], 1.5) #A1B1
    neur2 = McCulloch_Pitts_neuron([0, 1, 0, 1], 1.5) #A0B0
    neur3 = McCulloch_Pitts_neuron([1, 1], 1.5) #A1B1A0B0

    z1 = neur1.model(np.array(input))
```

```

z2 = neur2.model(np.array(input))
z3 = neur3.model(np.array([z1, z2]))

return list([z1,z2,z3])

```

:Y2 تعریف

```

def Y2(input):
    neur1 = McCulloch_Pitts_neuron([1, 0 , 1 , 0] , 1.5) #A1B1
    neur4 = McCulloch_Pitts_neuron([0, -1 , 0 , 0] , 0) #NOT A0 = A0'
    neur5 = McCulloch_Pitts_neuron([0, 0 , 0 , -1] , 0) #NOT B0 = B0'
    neur6 = McCulloch_Pitts_neuron([1, 1], 1) #A0'+B0'
    neur3 = McCulloch_Pitts_neuron([1, 1], 1.5) #A1B1(A0'+B0')

    z1 = neur1.model(np.array(input))
    z4 = neur4.model(np.array(input))
    z5 = neur5.model(np.array(input))
    z6 = neur6.model(np.array([z4, z5]))
    z3 = neur3.model(np.array([z1, z6]))

    return list([z1,z4,z5,z6,z3])

```

:Y1 تعریف

```

def Y1(input):
    neur7 = McCulloch_Pitts_neuron([1, 0 , 0 , 1] , 1.5) #A1B0
    neur8 = McCulloch_Pitts_neuron([0, 1 , 1 , 0] , 1.5) #A0B1
    neur6 = McCulloch_Pitts_neuron([1, 1], 1) # or gate
    neur9 = McCulloch_Pitts_neuron([-1, -1], -1.5) # not and gate
    neur3 = McCulloch_Pitts_neuron([1, 1], 1.5) # and gate -> output is:
A1B0 XOR A0B1

    z7 = neur7.model(np.array(input))
    z8 = neur8.model(np.array(input))
    z6 = neur6.model(np.array([z7, z8]))
    z9 = neur9.model(np.array([z7, z8]))
    z3 = neur3.model(np.array([z6, z9]))

    return list([z7,z8,z6,z9,z3])

```

:Y0 تعریف

```

def Y0(input):
    neur2 = McCulloch_Pitts_neuron([0, 1 , 0 , 1] , 1.5) #A0B0

    z2 = neur2.model(np.array(input))

    return list([z2])

```

مشاهده می‌شود که در مجموع با ۹ نورون این ضرب‌کننده ساخته شد.

حال خروجی  $Y$  را بررسی می‌کنیم که همان  $Y = Y_3Y_2Y_1Y_0$  می‌باشد:

```

X = [[0,0,0,0], [0,0,0,1], [0,0,1,0], [0,0,1,1],
      [0,1,0,0], [0,1,0,1], [0,1,1,0], [0,1,1,1],
      [1,0,0,0], [1,0,0,1], [1,0,1,0], [1,0,1,1],
      [1,1,0,0], [1,1,0,1], [1,1,1,0], [1,1,1,1]]

for i in X:
    res3 = Y3(i)
    res2 = Y2(i)
    res1 = Y1(i)
    res0 = Y0(i)

    print("Y with input as", str(i[0]) + " " + str(i[1]) + " " + str(i[2]) +
" " + str(i[3]), "goes to output ", str(res3[2]), str(res2[4]),
str(res1[4]), str(res0[0]))

```

این کد برگرفته از کد قسمت XNOR است با این تفاوت که ما در اینجا ۴ تا ورودی و ۴ تا خروجی داریم.

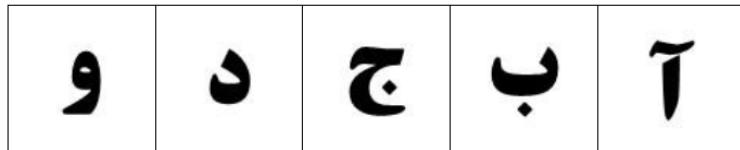
مشاهده خروجی:

```
Y with input as 0 0 0 0 goes to output 0 0 0 0
Y with input as 0 0 0 1 goes to output 0 0 0 0
Y with input as 0 0 1 0 goes to output 0 0 0 0
Y with input as 0 0 1 1 goes to output 0 0 0 0
Y with input as 0 1 0 0 goes to output 0 0 0 0
Y with input as 0 1 0 1 goes to output 0 0 0 1
Y with input as 0 1 1 0 goes to output 0 0 1 0
Y with input as 0 1 1 1 goes to output 0 0 1 1
Y with input as 1 0 0 0 goes to output 0 0 0 0
Y with input as 1 0 0 1 goes to output 0 0 1 0
Y with input as 1 0 1 0 goes to output 0 1 0 0
Y with input as 1 0 1 1 goes to output 0 1 1 0
Y with input as 1 1 0 0 goes to output 0 0 0 0
Y with input as 1 1 0 1 goes to output 0 0 1 1
Y with input as 1 1 1 0 goes to output 0 1 1 0
Y with input as 1 1 1 1 goes to output 1 0 0 1
```

همانطور که می بینیم این مقادیر با مقادیر بدست آمده از جدول حقیقت یکسان می باشد. پس کد ما درست کار می کند.

### سوال ۳

به این دفترچه کد مراجعه کنید و با اجرای سلول اول، ۵ داده تصویری مربوط به حروف الفبای فارسی که در شکل ۲ نشان داده شده است را دریافت کنید و سپس به سوالات زیر پاسخ دهید. دقت داشته باشید که در هر مرحله ارائه توضیحات متنی و دیداری مناسب لازم است. مثلاً می‌توانید ورودی نویزی و خروجی پیش‌بینی شده را در یک تصویر در کنار هم قرار دهید.



شکل ۲ : نمونه داده‌ها.

```
!gdown 1QTi7dTNAfFR5mG0rd8K3ZGvEIfSn_DS  
!unzip PersianData.zip
```

دستور بالا را اجرا می‌کنیم و ۵ داده تصویری را دریافت می‌کنیم:

```
Archive: PersianData.zip  
inflating: 1.jpg  
inflating: 2.jpg  
inflating: 3.jpg  
inflating: 4.jpg  
inflating: 5.jpg
```

#### قسمت ۱-۳

دوتابع پایتونی در سلول‌های دوم و سوم این دفترچه کد نوشته شده‌اند. اولین تابع تصویر را در ورودی خود دریافت و به صورت نمایش باینری درمی‌آورد و دومین تابع با افزودن نویز به داده‌ها، داده‌های جدید نویزی تولید می‌کند. در مورد نحوه عملکرد هریک از این توابع توضیح دهید. همچنان، می‌توانید این دستورات را به صورتی بهتر و کارآمدتر بازنویسی کنید.

بررسی تابع اول:

```
from PIL import Image, ImageDraw  
import random  
  
def convertImageToBinary(path) :  
    """  
    Convert an image to a binary representation based on pixel intensity.  
  
    Args:
```

```
    path (str): The file path to the input image.

>Returns:
    list: A binary representation of the image where white is
represented by -1 and black is represented by 1.
    """
    # Open the image file.
    image = Image.open(path)

    # Create a drawing tool for manipulating the image.
    draw = ImageDraw.Draw(image)

    # Determine the image's width and height in pixels.
    width = image.size[0]
    height = image.size[1]

    # Load pixel values for the image.
    pix = image.load()

    # Define a factor for intensity thresholding.
    factor = 100

    # Initialize an empty list to store the binary representation.
    binary_representation = []

    # Loop through all pixels in the image.
    for i in range(width):
        for j in range(height):
            # Extract the Red, Green, and Blue (RGB) values of the pixel.
            red = pix[i, j][0]
            green = pix[i, j][1]
            blue = pix[i, j][2]

            # Calculate the total intensity of the pixel.
            total_intensity = red + green + blue

            # Determine whether the pixel should be white or black based
            # on the intensity.
            if total_intensity > (((255 + factor) // 2) * 3):
                red, green, blue = 255, 255, 255 # White pixel
                binary_representation.append(-1)
            else:
                red, green, blue = 0, 0, 0 # Black pixel
                binary_representation.append(1)
```

```

# Set the pixel color accordingly.
draw.point((i, j), (red, green, blue))

# Clean up the drawing tool.
del draw

# Return the binary representation of the image.
return binary_representation

```

مشاهده می کنیم که تابع convertImageToBinary یک تصویر را به نمایش داده است و اساساً شدت پیکسل تبدیل می کند. این تابع از کتابخانه PIL برای تغییر تصویر استفاده می کند. در ادامه توضیحی مختصر از عملکرد تابع را به ترتیب سطرهای کد می دهیم:

۱. تابع مسیر فایل تصویر ورودی را به عنوان ورودی می گیرد.
  ۲. فایل تصویر را باز می کند و ابزاری برای تغییر تصویر ایجاد می کند.
  ۳. عرض و ارتفاع تصویر را به پیکسل مشخص می کند و مقادیر پیکسل را بارگیری می کند.
  ۴. سپس یک فاکتور برای آستانه گذاری شدت را تعريف می کند و یک لیست خالی برای ذخیره نمایش داده است.
  ۵. این تابع از تمام پیکسل های تصویر عبور می کند و شدت کلی هر پیکسل بر اساس مقادیر RGB آن محاسبه می کند.
  ۶. بر اساس شدت، تصمیم می گیرد که آیا پیکسل باید سفید یا مشکی باشد و مقدار متناظر (۰ - برای سفید، ۱ - برای مشکی) را به لیست نمایش داده می افزاید.
  ۷. رنگ پیکسل را مطابق با این شدت تنظیم می کند و نمایش داده می شود.
- به طور کلی دیدیم که این تابع به طور موثر تصویر را به نمایش داده است و مقدار متناظر (۰ - مشکی و ۱ - سفید) را به لیست نمایش داده می شود.

بررسی تابع دوم:

```

from PIL import Image, ImageDraw
import random

```

```
def generateNoisyImages():
    # List of image file paths
    image_paths = [
        "/content/1.jpg",
        "/content/2.jpg",
        "/content/3.jpg",
        "/content/4.jpg",
        "/content/5.jpg"
    ]

    for i, image_path in enumerate(image_paths, start=1):
        noisy_image_path = f"/content/noisy{i}.jpg"
        getNoisyBinaryImage(image_path, noisy_image_path)
        print(f"Noisy image for {image_path} generated and saved as {noisy_image_path}")

def getNoisyBinaryImage(input_path, output_path):
    """
    Add noise to an image and save it as a new file.

    Args:
        input_path (str): The file path to the input image.
        output_path (str): The file path to save the noisy image.
    """
    # Open the input image.
    image = Image.open(input_path)

    # Create a drawing tool for manipulating the image.
    draw = ImageDraw.Draw(image)

    # Determine the image's width and height in pixels.
    width = image.size[0]
    height = image.size[1]

    # Load pixel values for the image.
    pix = image.load()

    # Define a factor for introducing noise.
    noise_factor = 100

    # Loop through all pixels in the image.
    for i in range(width):
        for j in range(height):
            # Generate a random noise value within the specified factor.
            rand = random.randint(-noise_factor, noise_factor)
```

```

        # Add the noise to the Red, Green, and Blue (RGB) values of
the pixel.
        red = pix[i, j][0] + rand
        green = pix[i, j][1] + rand
        blue = pix[i, j][2] + rand

        # Ensure that RGB values stay within the valid range (0-255).
        if red < 0:
            red = 0
        if green < 0:
            green = 0
        if blue < 0:
            blue = 0
        if red > 255:
            red = 255
        if green > 255:
            green = 255
        if blue > 255:
            blue = 255

        # Set the pixel color accordingly.
        draw.point((i, j), (red, green, blue))

# Save the noisy image as a file.
image.save(output_path, "JPEG")

# Clean up the drawing tool.
del draw

# Generate noisy images and save them
generateNoisyImages()

```

دیدیم که تابع `getNoisyBinaryImage` تصویر ورودی را با افزودن نویز، تبدیل به تصویر نویزی می‌کند. این تابع نیز همانند تابع قبل از کتابخانه PIL برای تغییر تصویر استفاده می‌کند. ابتدا تصویر را باز می‌کند و سپس برای هر پیکسل، یک مقدار نویز تصادفی تولید می‌کند و آن را به مقادیر RGB پیکسل اضافه می‌کند. سپس مقادیر RGB را بررسی می‌کند تا در محدوده معتبر (۰-۲۵۵) باشند و در صورت لزوم، آن‌ها را تنظیم می‌کند. در نهایت، تصویر نویزی را به عنوان یک فایل جدید ذخیره می‌کند.

البته باید گفت معیار تنظیم نویز در این تابع noise\_factor می‌باشد که ما برای این سوال عدد ۱۰۰ را در نظر گرفتیم.

مشاهده خروجی:

```
Noisy image for /content/1.jpg generated and saved as /content/noisy1.jpg
Noisy image for /content/2.jpg generated and saved as /content/noisy2.jpg
Noisy image for /content/3.jpg generated and saved as /content/noisy3.jpg
Noisy image for /content/4.jpg generated and saved as /content/noisy4.jpg
Noisy image for /content/5.jpg generated and saved as /content/noisy5.jpg
```

حال که بهتر و کارآمدتر را مشاهده می‌کنیم:

```
from PIL import Image, ImageDraw
import random

def generateNoisyImages():
    # List of image file paths
    image_paths = [
        "/content/1.jpg",
        "/content/2.jpg",
        "/content/3.jpg",
        "/content/4.jpg",
        "/content/5.jpg"
    ]

    for i, image_path in enumerate(image_paths, start=1):
        noisy_image_path = f"/content/noisy{i}.jpg"
        getNoisyBinaryImage(image_path, noisy_image_path)
        print(f"Noisy image for {image_path} generated and saved as {noisy_image_path}")

def getNoisyBinaryImage(input_path, output_path):
    """
    Add noise and modify black points to white in an image and save it as
    a new file.

    Args:
        input_path (str): The file path to the input image.
        output_path (str): The file path to save the noisy image.
    """
    # Open the input image.
    image = Image.open(input_path)
```

```
# Create a drawing tool for manipulating the image.
draw = ImageDraw.Draw(image)

# Determine the image's width and height in pixels.
width = image.size[0]
height = image.size[1]

# Load pixel values for the image.
pix = image.load()

# Define a factor for introducing noise.
noise_factor = 1

# Loop through all pixels in the image.
for i in range(width):
    for j in range(height):
        # Generate a random noise value within the specified factor.
        rand = random.randint(-noise_factor, noise_factor)

        # Add the noise to the Red, Green, and Blue (RGB) values of
the pixel.
        red = pix[i, j][0] + rand
        green = pix[i, j][1] + rand
        blue = pix[i, j][2] + rand

        # Ensure that RGB values stay within the valid range (0-255).
        if red < 0:
            red = 0
        if green < 0:
            green = 0
        if blue < 0:
            blue = 0
        if red > 255:
            red = 255
        if green > 255:
            green = 255
        if blue > 255:
            blue = 255

        # Modify black points to white (you can adjust the threshold)
        if red < 50 and green < 50 and blue < 50:
            red = 255
            green = 255
            blue = 255
```

```

        # Set the pixel color accordingly.
        draw.point((i, j), (red, green, blue))

    # Save the noisy image as a file.
    image.save(output_path, "JPEG")

    # Clean up the drawing tool.
    del draw

# Generate noisy images and save them
generateNoisyImages()

```

تفاوت اصلی بین این تابع با تابع قبلی این است که تابع فقط نویز به تصویر اضافه می‌کند و تصویر را به صورت تصادفی تغییر می‌دهد، در حالی که این تابع علاوه بر اضافه کردن نویز، نقاط سیاه تصویر را به رنگ سفید تغییر می‌دهد. این کار با توجه به مقدار RGB نقاط مورد نظر انجام می‌شود.

### ۲-۳

یک شبکه عصبی (همینگ یا هاپفیلد) طراحی کنید که با اعمال ورودی دارای میزان مشخصی نویز برای هر یک از داده‌ها، خروجی متناسب با آن داده نویزی را بیابد. میزان نویز را تا حدی که شبکه شما ناموفق عمل کند افزایش دهید و نتایج را مقایسه و تحلیل کنید.

ما برای این سوال یک شبکه همینگ را انتخاب کردیم:

```

from pylab import *
from math import sqrt
import matplotlib.pyplot as plt
import os

# Define the path to the input image
IMAGE_PATH = "/content/noisy1.jpg"

def show(matrix):
    """
    Display a matrix in a formatted manner.

    Args:
        matrix (list of lists): The matrix to be displayed.
    """
    for j in range(len(matrix)):
        for i in range(len(matrix[0])):

```

```
        print("{:3f}".format(matrix[j][i]), end=" ")
        print(sep="")

def change(vector, a, b):
    """
    Transform a vector into a matrix of specified dimensions.

    Args:
        vector (list): The vector to be transformed.
        a (int): The number of columns in the resulting matrix.
        b (int): The number of rows in the resulting matrix.

    Returns:
        list of lists: The transformed matrix.
    """
    matrix = [[0 for j in range(a)] for i in range(b)]
    k = 0
    j = 0
    while k < b:
        i = 0
        while i < a:
            matrix[k][i] = vector[j]
            j += 1
            i += 1
        k += 1
    return matrix

def product(matrix, vector, T):
    """
    Multiply a matrix by a vector.

    Args:
        matrix (list of lists): The matrix to be multiplied.
        vector (list): The vector to be multiplied.
        T (float): The threshold parameter for the activation function.

    Returns:
        list: The resulting vector after multiplication.
    """
    result_vector = []
    for i in range(len(matrix)):
        x = 0
        for j in range(len(vector)):
            x = x + matrix[i][j] * vector[j]
        result_vector.append((x + T))
```

```
    return result_vector

def action(vector, T, Emax):
    """
    Activation function to process a vector.

    Args:
        vector (list): The input vector to be processed.
        T (float): The threshold parameter for the activation function.
        Emax (float): The maximum allowable value for the difference in
output vectors between consecutive iterations.

    Returns:
        list: The output vector after activation.
    """
    result_vector = []
    for value in vector:
        if value <= 0:
            result_vector.append(0)
        elif 0 < value <= T:
            result_vector.append(Emax * value)
        elif value > T:
            result_vector.append(T)
    return result_vector

def mysum(vector, j):
    """
    Calculate the sum of vector values excluding the element at index j.

    Args:
        vector (list): The input vector.
        j (int): The index of the element to be excluded from the sum.

    Returns:
        float: The sum of vector values with the element at index j
excluded.
    """
    p = 0
    total_sum = 0
    while p < len(vector):
        if p != j:
            total_sum = total_sum + vector[p]
        p += 1
    return total_sum
```

```
def norm(vector, p):
    """
    Calculate the difference between two vectors and compute the norm of
    the resulting vector.

    Args:
        vector (list): The first vector.
        p (list): The second vector for subtraction.

    Returns:
        float: The Euclidean norm of the difference between the two
        vectors.
    """
    difference = []
    for i in range(len(vector)):
        difference.append(vector[i] - p[i])
    sum = 0
    for element in difference:
        sum += element * element
    return sqrt(sum)

# List of paths to example images
path = [
    '/content/1.jpg',
    '/content/2.jpg',
    '/content/3.jpg',
    '/content/4.jpg',
    '/content/5.jpg',
]

x = [] # Binary representations of example images
print(os.path.basename(IMAGE_PATH))

# Convert and store binary representations of example images
for i in path:
    x.append(convertImageToBinary(i))

y = convertImageToBinary(IMAGE_PATH) # Binary representation of the input
image
entr = y
k = len(x) # Number of example images
a = 96 # Number of columns in the transformed matrix
b = 96 # Number of rows in the transformed matrix
entr = y
q = change(y, a, b) # Transformation of input image into a matrix
```

```

plt.matshow(q)
plt.colorbar()

m = len(x[0])
w = [[(x[i][j]) / 2 for j in range(m)] for i in range(k)] # Weight matrix
T = m / 2 # Activation function threshold parameter
e = round(1 / len(x), 1)
E = [[0 for j in range(k)] for i in range(k)] # Synaptic connection
matrix
Emax = 0.000001 # Maximum allowable difference norm between output
vectors in consecutive iterations
U = 1 / Emax

# Set values for the synaptic connection matrix
for i in range(k):
    for j in range(k):
        if j == i:
            E[i][j] = 1.0
        else:
            E[i][j] = -e

s = [product(w, y, T)] # Initial output vector
p = action(s[0], U, Emax)
y = [p]
i = 0
j = []
p = [0 for j in range(len(s[0]))]

# Iterate until the difference norm is less than Emax
while norm(y[i], p) >= Emax:
    s.append([0 for j in range(len(s[0]))])
    for j in range(len(s[0])):
        s[i + 1][j] = y[i][j] - e * mysum(y[i], j)
    y.append((action(s[i + 1], U, Emax)))
    i += 1
    p = y[i - 1]

print('Output Vectors Table:')
show(y)
print('Last Output Vector:', *y[len(y) - 1])

# Determine the class with the highest output value
result_index = y[len(y) - 1].index(max(y[len(y) - 1])) + 1

if max(y[len(y) - 1]) == 0:

```

```

print("The Hamming network cannot make a preference between classes.")
print("In the case of a small number of input characteristics, the
network may not be able to classify the image.")
plt.show()
exit()
else:
    q = change(x[result_index - 1], a, b)
    print('The highest positive output value is associated with class',
result_index)
    plt.matshow(q)
    plt.colorbar()
    plt.show()

```

به طور کلی می توان گفت که این شبکه همینگ برای طبقه بندی تصاویر و دسته بندی یک تصویر ورودی در میان مجموعه ای از تصاویر نمونه، کار می کند. حال به توضیح توابع این شبکه می پردازیم:

: show(matrix) نمایش یک ماتریس به صورت فرمت بندی شده.

: change(vector, a, b) تبدیل یک بردار به یک ماتریس با ابعاد مشخص.

: product(matrix, vector, T) ضرب یک ماتریس در یک بردار.

: action(vector, T, Emax) تابع فعال سازی برای پردازش یک بردار.

: mysum(vector, j) محاسبه مجموع مقادیر بردار با استثنای یک عنصر.

: norm(vector, p) محاسبه نرم اقلیدسی اختلاف بین دو بردار.

ویژگی های اصلی این توابع:

- نمونه های تصویری به نمایش دودویی تبدیل می شوند و در لیست  $x$  ذخیره می شوند
- تصویر ورودی هم به نمایش دودویی تبدیل شده و در  $y$  ذخیره می شود
- وزن های اولیه شبکه از مقادیر دودویی نمونه ها تقسیم بر دو بدست می آید
- ماتریس اتصالات جانبی  $E$  مهار رقابتی بین گره های خروجی را پیاده سازی می کند
- خروجی با ضرب وزن ها در ورودی و اعمال تابع فعال سازی بدست می آید
- خروجی تکراراً بر مبنای مهار جانبی به روزرسانی می شود تا همگرایی
- هنگام همگرایی، کلاس مربوط به گره بیشترین خروجی مثبت برگردانده می شود

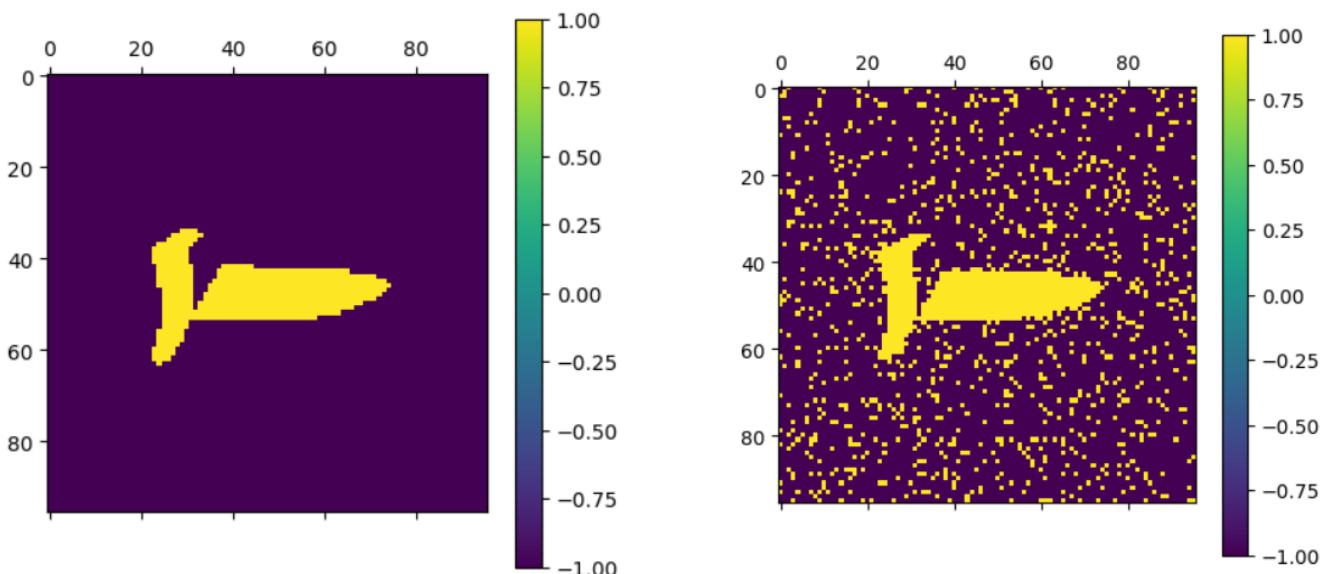
باید دقت شود که در این سوال  $noise\_factor = 100$  در نظر گرفته شده است.

بنابراین این شبکه همینگ از نمونه‌های آموزشی وزن‌ها را یادمی‌گیرد و با رقابت جانبی، تصویر ورودی را طبقه‌بندی می‌کند. به طور کلی هدف این شبکه، تعیین کلاس بر اساس بالاترین مقدار خروجی از شبکه همینگ می‌باشد.

مشاهده خروجی:

```
noisy1.jpg
Output Vectors Table:
0.008177 0.007412 0.007459 0.007650 0.007745
0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
Last Output Vector: 1.010840000000009e-15 0 0 2.5196e-16 3.88759999999994e-16
The highest positive output value is associated with class 1
```

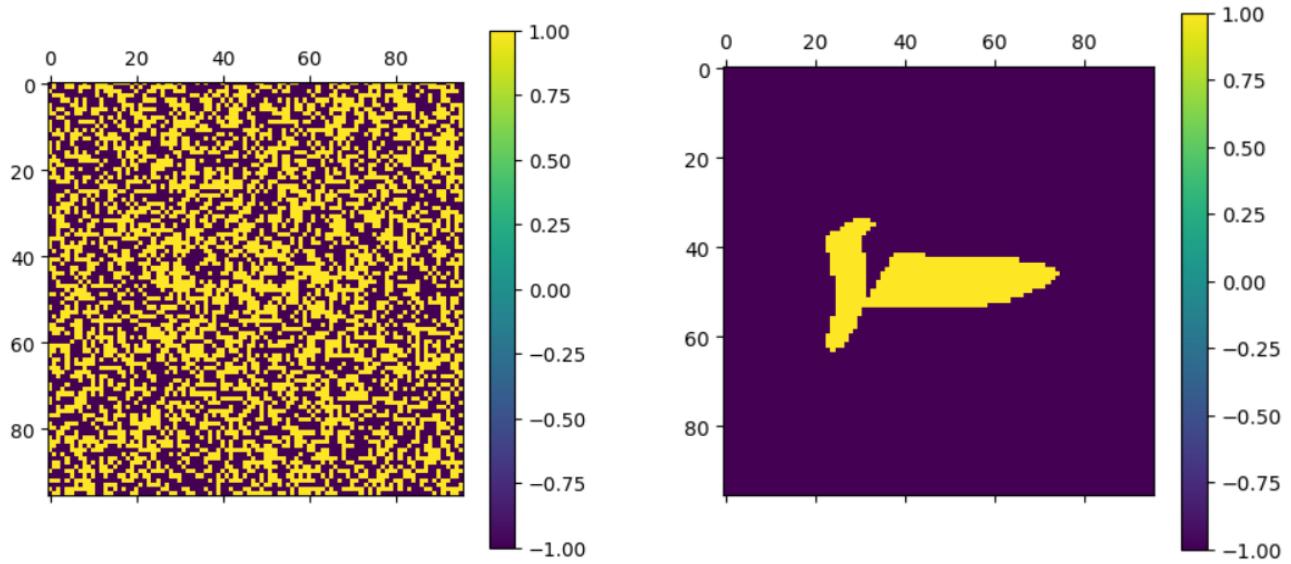
مشاهده تصویر دارای نویز و تصویر اصلی:



حال  $noise\_factor$  را افزایش می‌دهیم و برابر با  $1000$  قرار می‌دهیم و خروجی را مشاهده می‌کنیم:

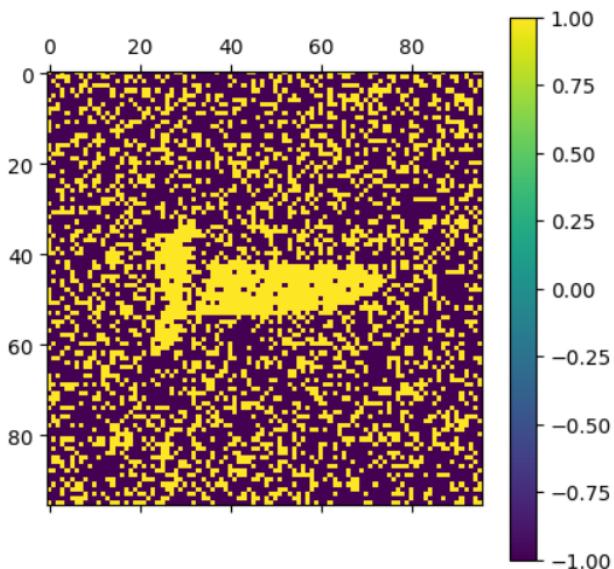
```
noisy1.jpg
Output Vectors Table:
0.005008 0.004893 0.004922 0.004935 0.004918
0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
Last Output Vector: 3.02239999999987e-16 1.36639999999974e-16 1.784000000000042e-16 1.971200000000065e-16 1.726400000000045e-16
The highest positive output value is associated with class 1
```

مشاهده تصویر دارای نویز و تصویر اصلی:

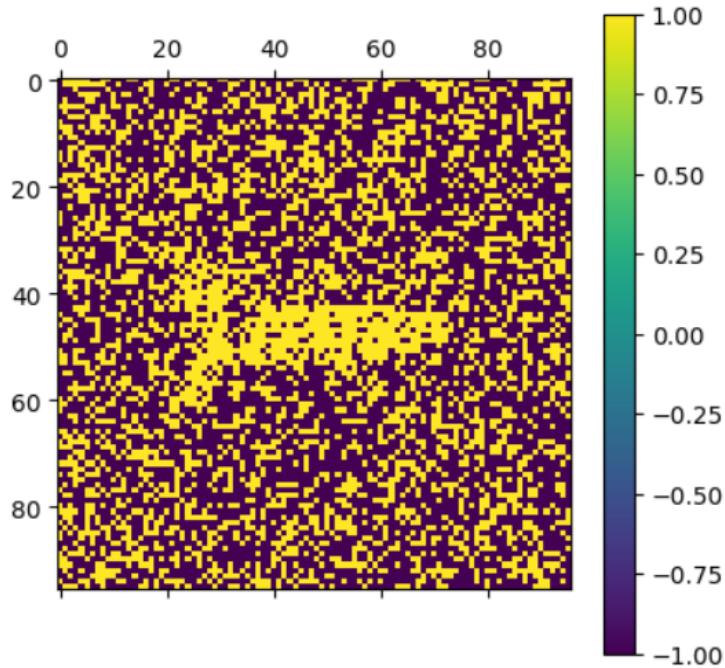


مشاهده می شود که شبکه ناموفق عمل می کند پس نویز بالا می باشد. پس حداکثر نویزی که می توان به شبکه داد مقداری بین ۱۰۰ و ۱۰۰۰ است که با آزمون و خطا می توان این عدد را پیدا کرد:

مشاهده تصویر دارای نویز به ازای  $\text{noise\_factor} = 200$



مشاهده تصویر دارای نویز به ازای  $\text{noise\_factor} = 300$



پس می‌توان گفت تقریباً به ازای نویز  $30\%$  شبکه ما به خوبی عمل نمی‌کند و مناسب نمی‌باشد.

همچنین برای مشاهده تغییرات بردار خروجی و وزن‌ها در هر مرحله از شبیه‌سازی می‌توان بدین صورت عمل کرد:

```
print('Output Vectors Table:')
for idx, output_vector in enumerate(y):
    print(f'Iteration {idx + 1}:', *output_vector)
    print('Weights (x, y):')
    for j in range(len(x[0])):
        print(f'x{j}: {w[0][j]:.3f}, y{j}: {w[1][j]:.3f}')
    print()
```

با این کدها جدول بردارهای خروجی شبکه در هر تکرار نمایش داده می‌شود. در هر تکرار ابتدا بردار خروجی آن تکرار چاپ می‌شود و سپس وزن‌های بین هر گره ورودی و گره‌های  $x$  و  $y$  نمایش داده می‌شوند. با این روش می‌توان رفتار شبکه در طول شبیه‌سازی را رصد کرده و اگر لازم باشد، الگوریتم یا مقادیر پارامترها را بهینه کرد.

مشاهده قسمتی از خروجی:

```
x4242: -0.500, y4242: -0.500
x4243: -0.500, y4243: -0.500
x4244: -0.500, y4244: -0.500
x4245: -0.500, y4245: -0.500
x4246: -0.500, y4246: -0.500
x4247: -0.500, y4247: -0.500
x4248: 0.500, y4248: -0.500
x4249: 0.500, y4249: -0.500
x4250: 0.500, y4250: -0.500
x4251: 0.500, y4251: -0.500
x4252: 0.500, y4252: -0.500
x4253: 0.500, y4253: -0.500
```

در ضمن می‌توان با دستور زیر مشخصات شکل از جمله شکل و ابعاد تصویر و تعداد ستون‌های تصویر را مشاهده کرد:

```
from PIL import Image

# Open the image
image_path = "/content/1.jpg"
image = Image.open(image_path)

# Get the shape (dimensions) and number of columns (width) of the image
image_shape = image.size
image_width = image_shape[0]

print("Image Shape (Dimensions):", image_shape)
print("Number of Columns (Width):", image_width)
```

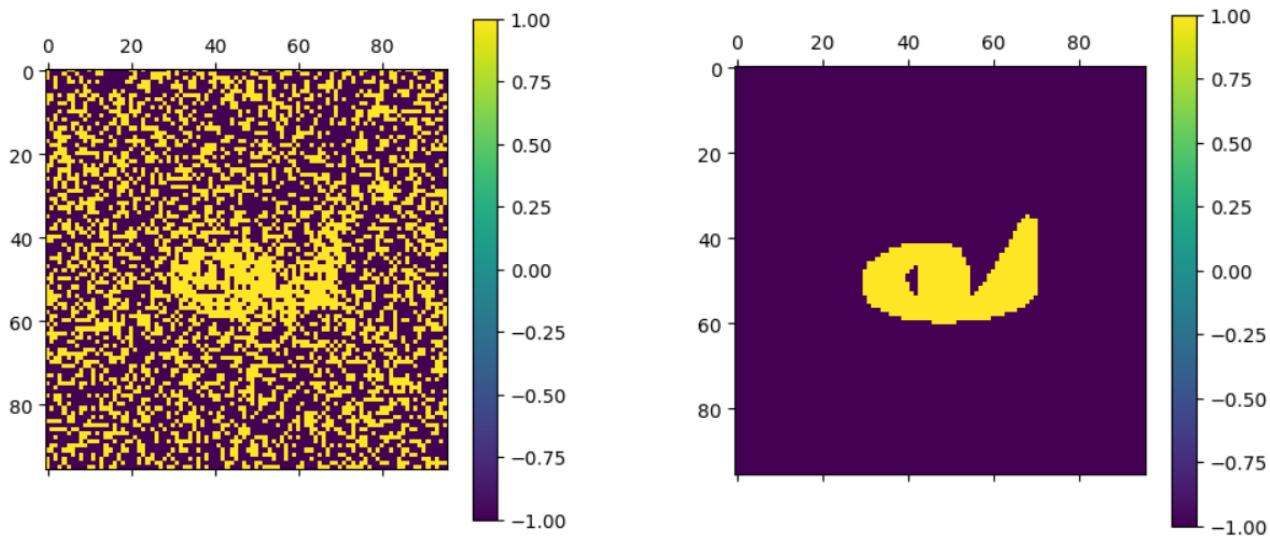
در این کد از (PIL) Python Image Library برای باز کردن تصویر استفاده می‌شود. در ادامه مسیر تصویر به صورت رشته در image\_path ذخیره می‌شود. سپس تابع open از PIL برای باز کردن تصویر از روی مسیر استفاده می‌شود.

مشاهده خروجی:

```
Image Shape (Dimensions): (96, 96)
Number of Columns (Width): 96
```

همین کارها را می‌توان روی ۴ تصویر دیگر انجام داد و نتیجه را مشاهده کرد.

در یک مثال دیگر خروجی شکل پنجم را مشاهده می‌کنیم:



مشاهده می‌شود که این شبکه با نویز ۳۰٪ برای هیچ کدام از شکل‌ها به درستی کار نمی‌کند.

### ۳-۳ قسمت

با الهام‌گرفتن ازتابع نوشته شده برای تولید داده‌های نویزی، یک تابع بنویسید که از داده‌های ورودی، خروجی‌های دارای Missing Point تولید کند. سپس عملکرد شبکه خود را با مقدار مشخصی Missing Point آزمایش و تحلیل کنید. اگر میزان Missing Point از چه حدی بیشتر شود عملکرد شبکه طراحی شده شما دچار اختلال می‌شود؟ راه حل چیست؟ (راهنمایی: نمونه داده دارای Missing Point در [شکل ۳](#) نشان داده شده است).



شکل ۳: نمونه داده دارای Missing Point

تابعی که خروجی‌های Missing Point تولید می‌کند به شرح زیر است:

```
from PIL import Image, ImageDraw
import random
```

```
def getNoisyBinaryImage(input_path, output_path, num_missing_points,
conversion_percentage):
    """
    Add noise to an image, generate missing points, and save it as a new
    file.

    Args:
        input_path (str): The file path to the input image.
        output_path (str): The file path to save the noisy image.
        num_missing_points (int): The number of missing points to
        generate.
        conversion_percentage (float): The percentage of black pixels to
        convert to white.
    """
    # Open the input image.
    image = Image.open(input_path)

    # Create a drawing tool for manipulating the image.
    draw = ImageDraw.Draw(image)

    # Determine the image's width and height in pixels.
    width = image.size[0]
    height = image.size[1]

    # Load pixel values for the image.
    pix = image.load()

    # Define a factor for introducing noise.
    noise_factor = 5

    # Loop through all pixels in the image.
    for i in range(width):
        for j in range(height):
            # Generate a random noise value within the specified factor.
            rand = random.randint(-noise_factor, noise_factor)

            # Add the noise to the Red, Green, and Blue (RGB) values of
            # the pixel.
            red = pix[i, j][0] + rand
            green = pix[i, j][1] + rand
            blue = pix[i, j][2] + rand

            # Ensure that RGB values stay within the valid range (0-255).
            if red < 0:
```

```
        red = 0
        if green < 0:
            green = 0
        if blue < 0:
            blue = 0
        if red > 255:
            red = 255
        if green > 255:
            green = 255
        if blue > 255:
            blue = 255

        # Convert some black pixels to white based on the conversion
percentage.
        if (red, green, blue) == (0, 0, 0) and random.random() <
conversion_percentage:
            red, green, blue = 255, 255, 255

        # Set the pixel color accordingly.
        draw.point((i, j), (red, green, blue))

    # Generate missing points in the image.
    for _ in range(num_missing_points):
        x = random.randint(0, width - 1)
        y = random.randint(0, height - 1)
        draw.point((x, y), (255, 255, 255)) # Set the missing point to
white

    # Save the noisy image as a file.
    image.save(output_path, "JPEG")

    # Clean up the drawing tool.
    del draw

from PIL import Image, ImageDraw
import random

def generateNoisyImages():
    # List of image file paths
    image_paths = [
        "/content/1.jpg",
        "/content/2.jpg",
        "/content/3.jpg",
        "/content/4.jpg",
        "/content/5.jpg"
```

```

]

for i, image_path in enumerate(image_paths, start=1):
    noisy_image_path = f"/content/noisy{i}.jpg"
    # Specify the number of missing points and conversion percentage
here
    getNoisyBinaryImage(image_path, noisy_image_path,
num_missing_points=500, conversion_percentage=0.1)
    print(f"Noisy image for {image_path} generated and saved as
{noisy_image_path}")

# Generate noisy images with missing points and black-to-white conversion
generateNoisyImages()

```

این کد همانند تابع قبل یک تابع به نام `getNoisyBinaryImage` را ایجاد می‌کند که یک تصویر ورودی را با `Missing Point` و ذخیره آن به عنوان یک فایل جدید، تبدیل می‌کند. این تابع از کتابخانه `PIL` برای باز کردن تصویر و ایجاد ابزار نقاشی برای تغییر تصویر استفاده می‌کند. سپس برای هر پیکسل در تصویر، یک مقدار نویز تصادفی تولید می‌کند و آن را به مقادیر RGB پیکسل اضافه می‌کند. سپس مقادیر RGB را بررسی می‌کند تا در محدوده معتبر (۰-۲۵۵) باشند و در صورت لزوم، آنها را تنظیم می‌کند. همچنین بر اساس درصد تبدیل، برخی از پیکسل‌های سیاه به سفید تبدیل می‌شوند. سپس تصویر نویزی به عنوان یک فایل جدید ذخیره می‌شود.

همچنین یک تابع دیگر به نام `generateNoisyImages` وجود دارد که یک لیست از مسیرهای فایل تصویر را دارد و برای هر تصویر ورودی، یک تصویر نویزی تولید می‌کند و آن را با نام جدیدی ذخیره می‌کند. این کد برای افزایش داده‌های آموزشی در مسائل پردازش تصویر مفید است.

در ضمن در این قسمت از سوال ابتدا تعداد `Missing Point` ها برابر با ۵۰۰ می‌باشد.

مشاهده خروجی تابع:

```

Noisy image for /content/1.jpg generated and saved as /content/noisy1.jpg
Noisy image for /content/2.jpg generated and saved as /content/noisy2.jpg
Noisy image for /content/3.jpg generated and saved as /content/noisy3.jpg
Noisy image for /content/4.jpg generated and saved as /content/noisy4.jpg
Noisy image for /content/5.jpg generated and saved as /content/noisy5.jpg

```

سپس کدی که مربوط به شبکه عصبی همینگ است را وارد می‌کنیم:

```

from pylab import *

```

```
from math import sqrt
import matplotlib.pyplot as plt
import os

# Define the path to the input image
IMAGE_PATH = "/content/noisy1.jpg"

def show(matrix):
    """
    Display a matrix in a formatted manner.

    Args:
        matrix (list of lists): The matrix to be displayed.
    """
    for j in range(len(matrix)):
        for i in range(len(matrix[0])):
            print("{:3f}".format(matrix[j][i]), end=" ")
        print(sep="")

def change(vector, a, b):
    """
    Transform a vector into a matrix of specified dimensions.

    Args:
        vector (list): The vector to be transformed.
        a (int): The number of columns in the resulting matrix.
        b (int): The number of rows in the resulting matrix.

    Returns:
        list of lists: The transformed matrix.
    """
    matrix = [[0 for j in range(a)] for i in range(b)]
    k = 0
    j = 0
    while k < b:
        i = 0
        while i < a:
            matrix[k][i] = vector[j]
            j += 1
            i += 1
        k += 1
    return matrix

def product(matrix, vector, T):
    """
```

```
Multiply a matrix by a vector.

Args:
    matrix (list of lists): The matrix to be multiplied.
    vector (list): The vector to be multiplied.
    T (float): The threshold parameter for the activation function.

Returns:
    list: The resulting vector after multiplication.
"""

result_vector = []
for i in range(len(matrix)):
    x = 0
    for j in range(len(vector)):
        x = x + matrix[i][j] * vector[j]
    result_vector.append((x + T))
return result_vector

def action(vector, T, Emax):
    """
    Activation function to process a vector.

    Args:
        vector (list): The input vector to be processed.
        T (float): The threshold parameter for the activation function.
        Emax (float): The maximum allowable value for the difference in
output vectors between consecutive iterations.

    Returns:
        list: The output vector after activation.
    """

    result_vector = []
    for value in vector:
        if value <= 0:
            result_vector.append(0)
        elif 0 < value <= T:
            result_vector.append(Emax * value)
        elif value > T:
            result_vector.append(T)
    return result_vector

def mysum(vector, j):
    """
    Calculate the sum of vector values excluding the element at index j.

```

```

Args:
    vector (list): The input vector.
    j (int): The index of the element to be excluded from the sum.

Returns:
    float: The sum of vector values with the element at index j
excluded.

"""
p = 0
total_sum = 0
while p < len(vector):
    if p != j:
        total_sum = total_sum + vector[p]
    p += 1
return total_sum

def norm(vector, p):
    """
    Calculate the difference between two vectors and compute the norm of
    the resulting vector.

    Args:
        vector (list): The first vector.
        p (list): The second vector for subtraction.

    Returns:
        float: The Euclidean norm of the difference between the two
vectors.

    """
difference = []
for i in range(len(vector)):
    difference.append(vector[i] - p[i])
sum = 0
for element in difference:
    sum += element * element
return sqrt(sum)

# List of paths to example images
path = [
    '/content/1.jpg',
    '/content/2.jpg',
    '/content/3.jpg',
    '/content/4.jpg',
    '/content/5.jpg',
]

```

```

x = [] # Binary representations of example images
print(os.path.basename(IMAGE_PATH))

# Convert and store binary representations of example images
for i in path:
    x.append(convertImageToBinary(i))

y = convertImageToBinary(IMAGE_PATH) # Binary representation of the input
image
entr = y
k = len(x) # Number of example images
a = 96 # Number of columns in the transformed matrix
b = 96 # Number of rows in the transformed matrix
entr = y
q = change(y, a, b) # Transformation of input image into a matrix
plt.matshow(q)
plt.colorbar()

m = len(x[0])
w = [[(x[i][j]) / 2 for j in range(m)] for i in range(k)] # Weight matrix
T = m / 2 # Activation function threshold parameter
e = round(1 / len(x), 1)
E = [[0 for j in range(k)] for i in range(k)] # Synaptic connection
matrix
Emax = 0.000001 # Maximum allowable difference norm between output
vectors in consecutive iterations
U = 1 / Emax

# Set values for the synaptic connection matrix
for i in range(k):
    for j in range(k):
        if j == i:
            E[i][j] = 1.0
        else:
            E[i][j] = -e

s = [product(w, y, T)] # Initial output vector
p = action(s[0], U, Emax)
y = [p]
i = 0
j = []
p = [0 for j in range(len(s[0]))]

# Iterate until the difference norm is less than Emax

```

```

while norm(y[i], p) >= Emax:
    s.append([0 for j in range(len(s[0]))])
    for j in range(len(s[0])):
        s[i + 1][j] = y[i][j] - e * mysum(y[i], j)
    y.append((action(s[i + 1], U, Emax)))
    i += 1
    p = y[i - 1]

print('Output Vectors Table:')
show(y)
print('Last Output Vector:', *y[len(y) - 1])

# Determine the class with the highest output value
result_index = y[len(y) - 1].index(max(y[len(y) - 1])) + 1

if max(y[len(y) - 1]) == 0:
    print("The Hamming network cannot make a preference between classes.")
    print("In the case of a small number of input characteristics, the network may not be able to classify the image.")
    plt.show()
    exit()
else:
    q = change(x[result_index - 1], a, b)
    print('The highest positive output value is associated with class',
          result_index)
    plt.matshow(q)
    plt.colorbar()
    plt.show()

```

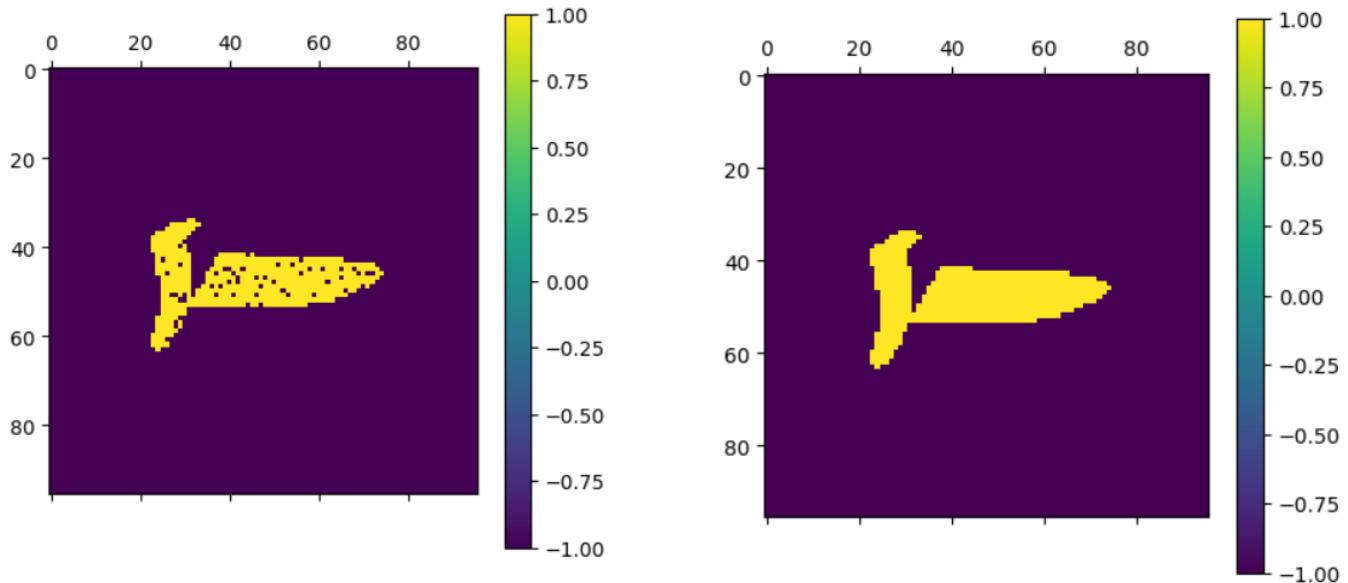
مشاهده خروجی:

```

noisy1.jpg
Output Vectors Table:
0.009157 0.008260 0.008267 0.008576 0.008643
0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
Last Output Vector: 1.173240000000001e-15 0 0 3.3659999999999984e-16 4.330799999999993e-16
The highest positive output value is associated with class 1

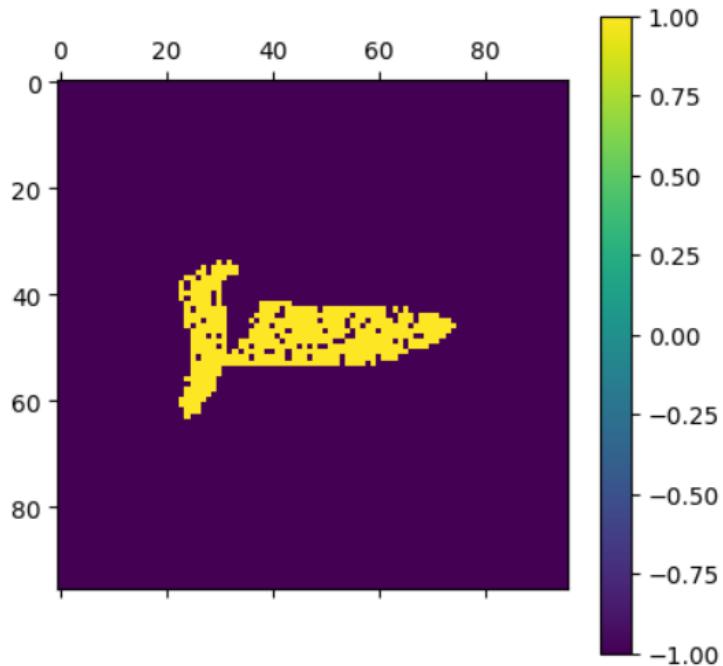
```

مشاهده شکل اصلی و شکل دارای Missing Point به ازای ۵۰۰:



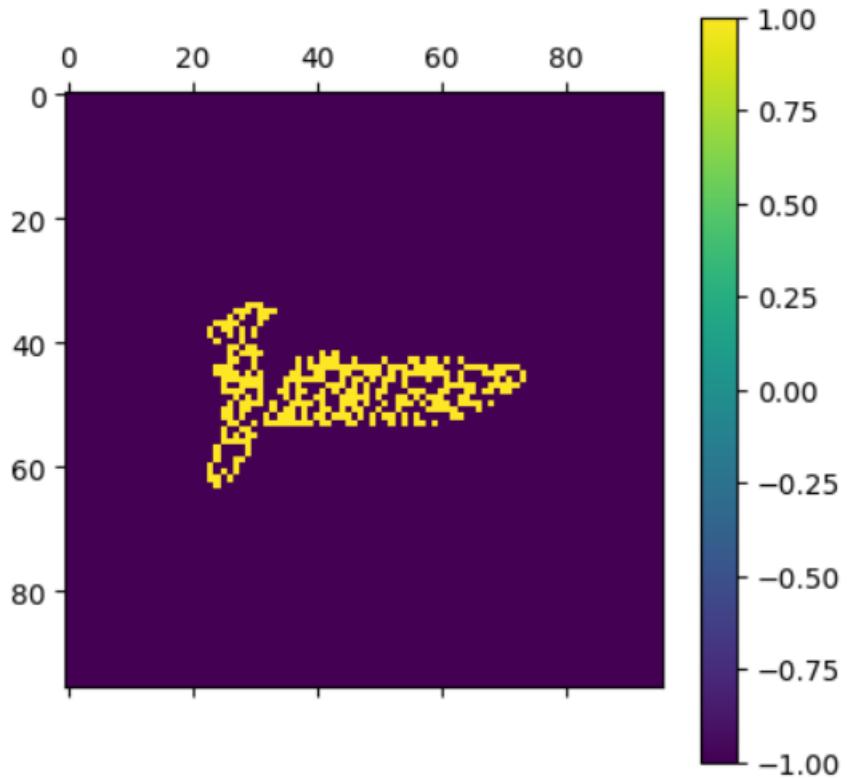
با این مقدار میزان عملکرد شبکه ما مناسب است. پس مقادیر بیشتر به Missing Point می‌دهیم:

مشاهده شکل دارای Missing Point به ازای ۱۰۰۰:



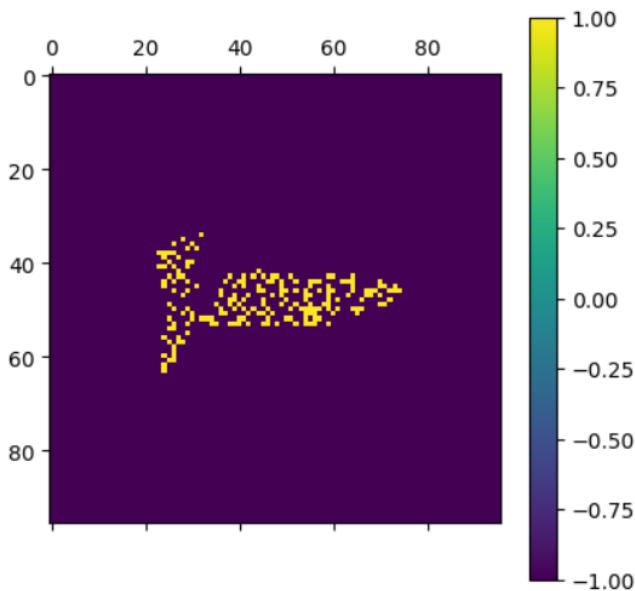
باز هم عملکرد شبکه قابل قبول می‌باشد.

مشاهده شکل داری Missing Point به ازای :۵۰۰۰



مشاهده می شود که به ازای ۵۰۰۰ آرام آرام شبکه ما توانایی خود را از دست می دهد.

مشاهده شکل داری Missing Point به ازای :۱۰۰۰۰



می‌بینیم که هر چه تعداد Missing Point زیاد می‌شود عملکرد شبکه همینگ ما ضعیفتر شده و شبکه ما دچار اختلال می‌شود. برای رفع مشکل عملکرد ضعیف شبکه عصبی همینگ با افزایش تعداد Missing Point ، می‌توان از روش‌های زیر استفاده کرد:

۱. استفاده از داده‌افزایی (Data Augmentation): با اعمال تغییرات تصادفی به تصاویر، مانند چرخش، تغییر اندازه، و شیفت دادن، می‌توان تنوع داده‌ها را افزایش داد. این کار می‌تواند به شبکه عصبی کمک کند تا الگوهای مختلفی را یاد بگیرد و در برابر داده‌های جدید و ناهمگن مقاوم‌تر شود.

۲. استفاده از شبکه‌های مولد (Generative Adversarial Networks - GANs): شبکه‌های مولد می‌توانند برای تولید داده‌های جدید و مصنوعی که شبیه به داده‌های واقعی هستند، استفاده شوند. این داده‌های تولید شده می‌توانند به عنوان داده‌های آموزشی برای شبکه عصبی همینگ استفاده شوند.

۳. استفاده از شبکه‌های مولد تصویر (Image Generative Networks): این شبکه‌ها می‌توانند برای تولید تصاویر مصنوعی با کیفیت بالا و شبیه به داده‌های واقعی استفاده شوند. این تصاویر می‌توانند به عنوان داده‌های آموزشی برای شبکه عصبی همینگ استفاده شوند.

۴. استفاده از شبکه‌های بازگردنی (Recurrent Neural Networks - RNNs): این شبکه‌ها می‌توانند برای مدل‌سازی داده‌های دنباله‌ای مانند تصاویر استفاده شوند. با استفاده از این شبکه‌ها می‌توان الگوهای زمانی و مکانی در داده‌ها را بهتر یاد گرفت.

۵. استفاده از شبکه‌های مولد تصویر مبتنی بر توجه (Attention-based Image Generative Networks): این شبکه‌ها می‌توانند با استفاده از مکانیزم توجه، دقیق در تولید تصاویر را افزایش دهند.

## ۴ سوال

یک مجموعه داده برای پیش‌بینی قیمت خانه‌ها را از طریق این پیوند دانلود کنید و مراحل ذکر شده در سوالات بعدی را برای فایل data.csv آن انجام دهید. لازم است که هر قسمت و مورد خواسته شده را با استفاده از دستورات پایتون انجام دهید و در جاهایی که نیاز است، نتایج را به صورت دقیق و کامل نمایش داده و تحلیل کنید.

ابتدا کتابخانه‌های مورد نیاز را می‌نویسیم:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import warnings
warnings.filterwarnings("ignore")
from sklearn.metrics import r2_score
from keras.models import Sequential
from keras.layers import Dense
```

توضیحات مورد نیاز کتابخانه‌ها:

۱. pandas: این کتابخانه برای کار با داده‌های جدولی و سری‌های زمانی استفاده می‌شود. این کتابخانه قدرتمند برای تحلیل و تغییر داده‌هاست.
۲. matplotlib.pyplot: این کتابخانه برای تصویرسازی داده‌ها استفاده می‌شود. این ابزار به شما امکان می‌دهد تا نمودارها و گراف‌های مختلف را برای نمایش داده‌های خود ایجاد کنید.
۳. numpy: این کتابخانه محاسبات عددی را فراهم می‌کند. از آرایه‌ها، ماتریس‌ها و توابع ریاضی پیشرفته برای انجام عملیات محاسباتی استفاده می‌کند.
۴. sklearn.preprocessing: این کتابخانه برای پیش‌پردازش داده‌ها، مانند مقیاس‌بندی و تبدیل داده‌ها به فرم قابل قبول برای مدل‌های یادگیری ماشین، استفاده می‌شود.
۵. seaborn: این کتابخانه برای تصویرسازی داده‌ها و رسم نمودارهای زیبا و اطلاعاتی استفاده می‌شود.

۶. `sklearn.model_selection`: این کتابخانه برای تقسیم داده‌ها به داده‌های آموزش و آزمون برای ارزیابی مدل‌های یادگیری ماشین استفاده می‌شود.

۷. `warnings`: این کتابخانه برای مدیریت هشدارها و اخطارهای مربوط به کدها استفاده می‌شود.

۸. `sklearn.metrics`: این کتابخانه برای اندازه‌گیری عملکرد مدل‌های یادگیری ماشین، مانند معیارهای دقت و دیگر معیارهای ارزیابی، استفاده می‌شود.

۹. `keras.layers` و `keras.models`: این کتابخانه‌ها برای ایجاد و ساختاردهی مدل‌های شبکه‌های عصبی و لایه‌های مختلف آن‌ها استفاده می‌شود. Keras یک کتابخانه محبوب برای طراحی و آموزش مدل‌های عصبی است.

## ۱-۴ قسمت

فایل CSV مربوط به این سوال را خوانده و سپس تابع `Info` از Pandas فراخوانی کنید. تعداد داده‌هایی که `Nan` هستند را بر حسب هر ستون نمایش دهید و اگر نیاز است دستوراتی برای رفع این مشکل بنویسید.

ابتدا فایل مورد نظر را با لینک موجود در گوگل درایو به پایتون فراخوانی می‌کنیم:

```
# part 1
#https://drive.google.com/file/d/1LjgD0GCoCoi_nIXp4AkRsSyvd8CNd1hY1/view?usp=sharing
!gdown 1LjgD0GCoCoi_nIXp4AkRsSyvd8CNd1hY1
```

```
data = pd.read_csv('/content/data.csv')
data.info()
```

با استفاده از تابع `read_csv` از کتابخانه `pandas` فایل CSV را می‌خوانیم و آن را در متغیر `data` ذخیره می‌کنیم. مسیر فایل CSV نیز به عنوان آرگومان به تابع داده شده است.

سپس با استفاده از تابع `info` اطلاعات مربوط به داده‌های موجود در فایل CSV را نمایش می‌دهیم. این اطلاعات شامل تعداد سطرها، تعداد ستون‌ها، نام ستون‌ها و نوع داده‌های هر ستون است.

حال می‌توان خروجی را مشاهده کرد:

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4600 entries, 0 to 4599
Data columns (total 18 columns):
 #   Column            Non-Null Count Dtype  
--- 
 0   date              4600 non-null   object  
 1   price             4600 non-null   float64 
 2   bedrooms          4600 non-null   float64 
 3   bathrooms          4600 non-null   float64 
 4   sqft_living        4600 non-null   int64   
 5   sqft_lot           4600 non-null   int64   
 6   floors             4600 non-null   float64 
 7   waterfront         4600 non-null   int64   
 8   view               4600 non-null   int64   
 9   condition          4600 non-null   int64   
 10  sqft_above         4600 non-null   int64   
 11  sqft_basement      4600 non-null   int64   
 12  yr_built           4600 non-null   int64   
 13  yr_renovated       4600 non-null   int64   
 14  street              4600 non-null   object  
 15  city                4600 non-null   object  
 16  statezip            4600 non-null   object  
 17  country              4600 non-null   object  
dtypes: float64(4), int64(9), object(5)
memory usage: 647.0+ KB

```

همانطور که مشاهده می شود همه ستون های خانه هستند در خروجی `data.info()` مشاهده می شود.

در ادامه بد نیست نگاهی به خود فایل `data` بیاندازیم:

```
data
```

مشاهده خروجی:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	street	city	statezip	country
0	2014-05-02 00:00:00	3.130000e+05	3.0	1.50	1340	7912	1.5	0	0	3	1340	0	1955	2005	18810 Densmore Ave N	Shoreline	WA 98133	USA
1	2014-05-02 00:00:00	2.384000e+06	5.0	2.50	3650	9050	2.0	0	4	5	3370	280	1921	0	709 W Blaine St	Seattle	WA 98119	USA
2	2014-05-02 00:00:00	3.420000e+05	3.0	2.00	1930	11947	1.0	0	0	4	1930	0	1966	0	26206-26214 143rd Ave SE	Kent	WA 98042	USA
3	2014-05-02 00:00:00	4.200000e+05	3.0	2.25	2000	8030	1.0	0	0	4	1000	1000	1963	0	857 170th Pl NE	Bellevue	WA 98006	USA
4	2014-05-02 00:00:00	5.500000e+05	4.0	2.50	1940	10500	1.0	0	0	4	1140	800	1976	1992	9105 170th Ave NE	Redmond	WA 98052	USA
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4595	2014-07-09 00:00:00	3.081667e+05	3.0	1.75	1510	6360	1.0	0	0	4	1510	0	1954	1979	501 N 143rd St	Seattle	WA 98133	USA
4596	2014-07-09 00:00:00	5.343333e+05	3.0	2.50	1460	7573	2.0	0	0	3	1460	0	1983	2009	14855 SE 10th Pl	Bellevue	WA 98007	USA
4597	2014-07-09 00:00:00	4.169042e+05	3.0	2.50	3010	7014	2.0	0	0	3	3010	0	2009	0	759 Ilwaco Pl NE	Renton	WA 98059	USA
4598	2014-07-10 00:00:00	2.034000e+05	4.0	2.00	2090	6630	1.0	0	0	3	1070	1020	1974	0	5148 S Creston St	Seattle	WA 98178	USA
4599	2014-07-10 00:00:00	2.206000e+05	3.0	2.50	1490	8102	2.0	0	0	4	1490	0	1990	0	16717 SE 258th St	Covington	WA 98042	USA

4600 rows x 18 columns

در ادامه تعداد داده‌هایی که Nan هستند را بر حسب هر ستون نمایش می‌دهیم:

```
nan = data.isnull().sum()  
nan
```

بررسی کد:

: این دستور برای تشخیص مقادیر نال (NaN) در هر سلول داده‌ها استفاده می‌شود. این دستور یک DataFrame جدید ایجاد می‌کند که مقادیر آن True یا False است، به‌طوری که True نشان‌دهنده مقادیر نال است.

: این دستور تعداد مقادیر نال در هر ستون را محاسبه می‌کند. با استفاده از تابع sum() هر ستون اعداد True که مقادیر نال را نشان می‌دهند را جمع می‌زنیم و تعداد مقادیر نال در هر ستون را بدست می‌آوریم.

مشاهده خروجی و تعداد مقادیر نال در هر ستون :

date	0
price	0
bedrooms	0
bathrooms	0
sqft_living	0
sqft_lot	0
floors	0
waterfront	0
view	0
condition	0
sqft_above	0
sqft_basement	0
yr_built	0
yr_renovated	0
street	0
city	0
statezip	0
country	0
dtype:	int64

برای رفع این مشکل می‌توان از دو روش زیر استفاده کرد:

(۱)

```
data.fillna(data.mean(), inplace = True )
```

این دستور برای پر کردن مقادیر نال در داده‌ها با میانگین مقادیر هر ستون استفاده می‌شود. آرگومان `inplace=True` نشان می‌دهد که تغییرات باید در متغیر اصلی `data` اعمال شود. این کار معمولاً برای پر کردن مقادیر نال با مقادیر معقول و بهینه استفاده می‌شود.

(۲)

```
data.dropna(inplace = True)
```

این دستور برای حذف تمام ردیف‌های دارای مقدار نال در داده‌ها استفاده می‌شود. این کار معمولاً برای حذف ردیف‌هایی که دارای مقادیر نال هستند و در تحلیل داده‌ها معنی ندارند، استفاده می‌شود.

## ۲-۴

ماتریس همبستگی را رسم کنید. چه ویژگی‌ای با قیمت همبستگی بیشتری دارد؟

```
# part 2
corr_mat = data.corr()
plt.figure(figsize=(10, 8))
plt.imshow(corr_mat, cmap='viridis', interpolation='none')
plt.colorbar()
plt.xticks(range(len(corr_mat)), corr_mat.columns, rotation=90)
plt.yticks(range(len(corr_mat)), corr_mat.columns)
plt.show()
```

این کد برای رسم نمودار heatmap از ماتریس همبستگی داده‌ها استفاده می‌شود. در قسمت زیر تک‌تک اجزای کد را بررسی می‌کنیم:

(`data.corr()`: این دستور ماتریس همبستگی داده‌ها را محاسبه می‌کند. این ماتریس نشان‌دهنده ارتباط بین متغیرها است.

(`plt.figure(figsize=(10, 8))`: این دستور یک شکل و `figure` جدید با اندازه مشخص ایجاد می‌کند.

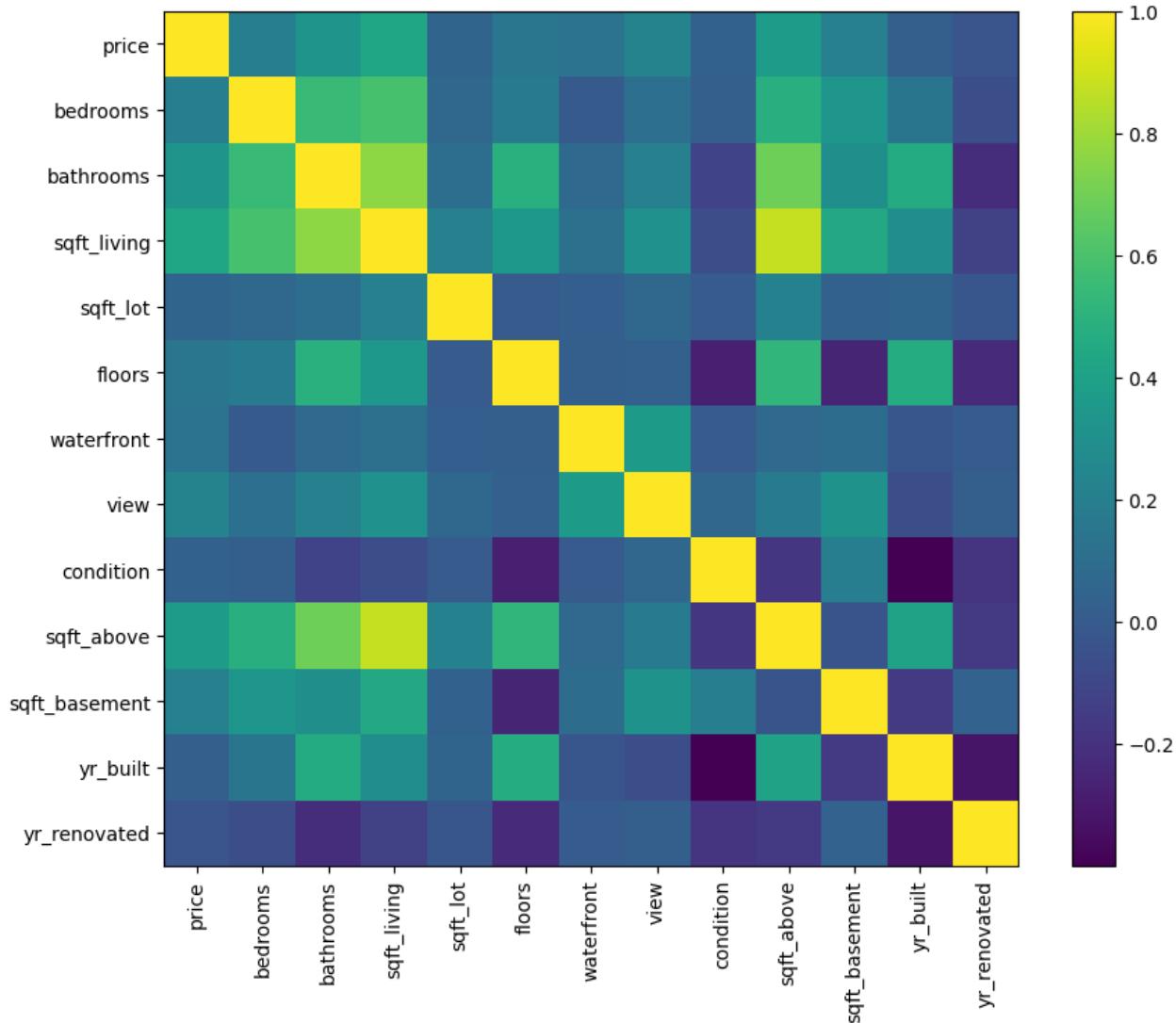
(`plt.imshow(corr_mat, cmap='viridis', interpolation='none')`: این دستور به `figure` اجازه می‌دهد تا ماتریس همبستگی را به صورت تصویر نمایش دهد. 'viridis' نشان می‌دهد که از رنگ‌های مختلف برای نمایش ارتباط‌ها استفاده شود.

(`plt.colorbar()`: این دستور نمودار رنگی را که به ماتریس همبستگی مربوط است نمایش می‌دهد.

: plt.yticks و plt.xticks این دستورها محورهای x و y نمودار را تنظیم می کنند.

: plt.show() این دستور نمودار را نمایش می دهد.

با اجرای این کد، یک نمودار heatmap از ماتریس همبستگی داده ها به تصویر کشیده می شود. این نمودار را مشاهده می کنیم:



در ضمن می توان این ماتریس را با دستور Seaborn رسم کرد که خروجی پر کاربردتری را به ما می دهد، زیرا در خروجی این دستور ما میزان همبستگی هر ویژگی را مشاهده می کنیم:

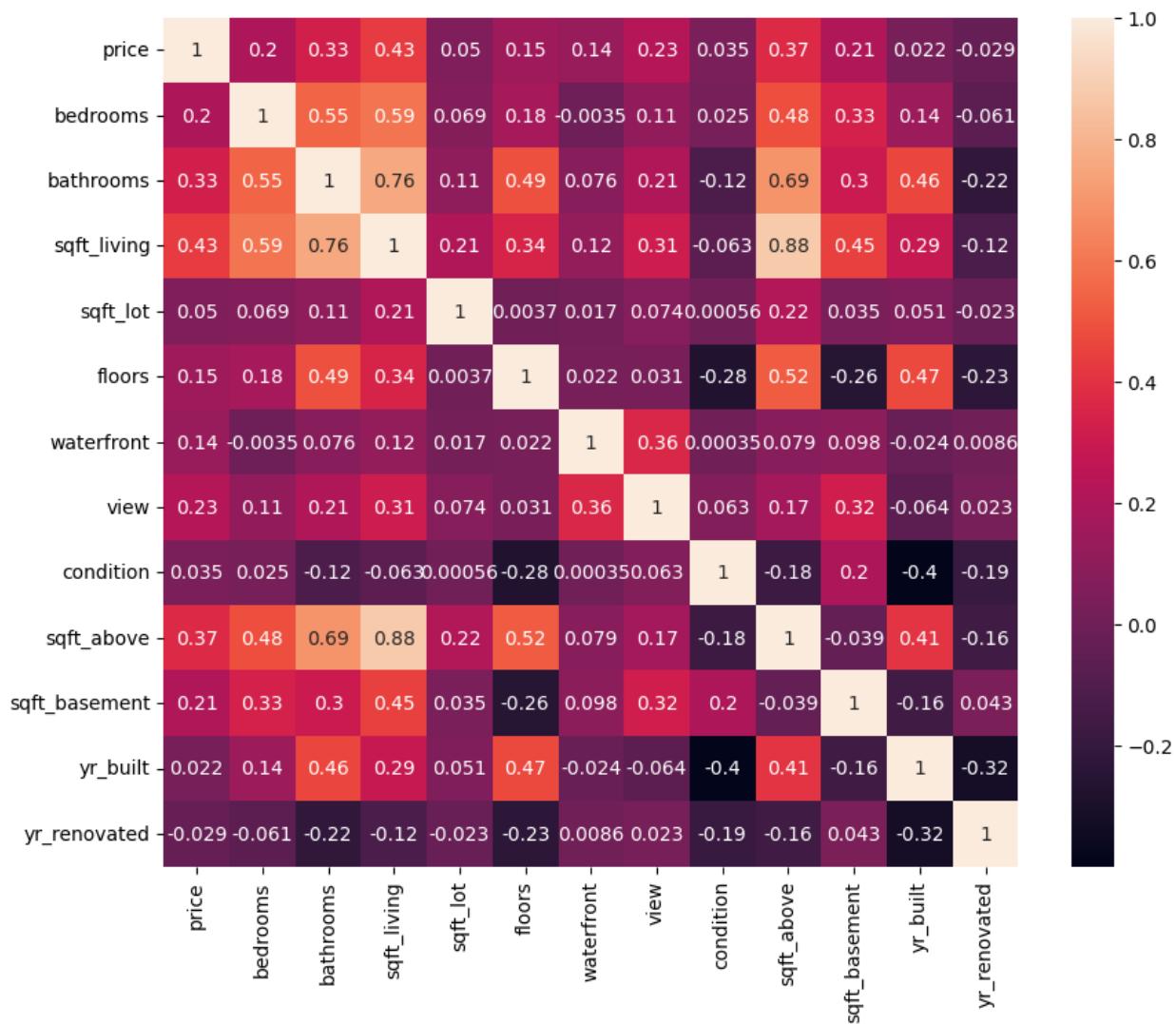
```
plt.figure(figsize=(10, 8))
sns.heatmap(corr_mat, annot=True)
plt.show()
```

توضیح کلی کد:

plt.figure(figsize=(10, 8))  
sns.heatmap(corr\_mat, annot=True)  
کتابخانه Seaborn رسم می‌کند. آرگومان 'annot=True' نشان می‌دهد که مقادیر واقعی ماتریس همبستگی روی نمودار نشان داده شوند.

با اجرای این کد، یک نمودار heatmap از ماتریس همبستگی داده‌ها به تصویر کشیده می‌شود، همانطور که گفتیم با استفاده از کتابخانه Seaborn می‌توان نمودار را با استایل‌های زیباتر و اطلاعات بیشتری نمایش داد.

مشاهده خروجی:



با توجه به نمودار بالا می‌توان گفت که ویژگی price با ویژگی sqft\_living بیشترین همبستگی را دارد و مقدار این همبستگی ۴۳٪ است. حال با استفاده از دستورات پایتونی می‌خواهیم این ویژگی و این مقدار را محاسبه کنیم: ابتدا تک تک ویژگی‌های این داده را با استفاده از دستور زیر مشاهده می‌کنیم:

```
num = data.select_dtypes(exclude=['object']).columns  
num
```

در خط اول با استفاده از تابع select\_dtypes() از کتابخانه pandas، ستون‌های مربوط به انواع داده‌های عددی (جز نوع داده object) را از دیتافریم data انتخاب می‌کند. سپس در خط دوم مقدار متغیر num که ویژگی‌های ما و شامل ستون‌های عددی انتخاب شده هستند، چاپ می‌شود.

مشاهده خروجی:

```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',  
       'waterfront', 'view', 'condition', 'sqft_above', 'sqft_basement',  
       'yr_built', 'yr_renovated'],  
      dtype='object')
```

در ادامه به مقایسه ویژگی‌ها می‌پردازیم:

```
max_corr_feature_with_price =  
corr_mat['price'].nlargest(2).iloc[1:2].index[0]  
max_corr_value_with_price =  
corr_mat['price'].nlargest(2).iloc[1:2].values[0]  
print(f"max_corr_feature_with_price: {max_corr_feature_with_price}  
max_corr_value_with_price: {max_corr_value_with_price}")
```

این کد برای پیدا کردن ویژگی با بیشترین همبستگی با ستون قیمت (price) از ماتریس همبستگی استفاده می‌شود. بررسی خطهای کد:

corr\_mat['price'].nlargest(2): دو ویژگی با بیشترین همبستگی مثبت با ستون 'price' را انتخاب می‌کند. تنها ویژگی دوم را انتخاب می‌کند (زیرا ویژگی اول که بیشترین همبستگی را با price دارد همان خود است، پس باید ویژگی دوم انتخاب شود).

: نام ویژگی مورد نظر را برمی گرداند.

: مقدار همبستگی آن ویژگی را برمی گرداند.

در نهایت نام ویژگی و مقدار همبستگی اش چاپ می شود:

```
max_corr_feature_with_price: sqft_living max_corr_value_with_price:  
0.4304100254326265
```

### ۳-۴ قسمت

نمودار توزیع قیمت و نمودار قیمت و ویژگی ای که همبستگی زیادی با قیمت دارد را رسم کنید.

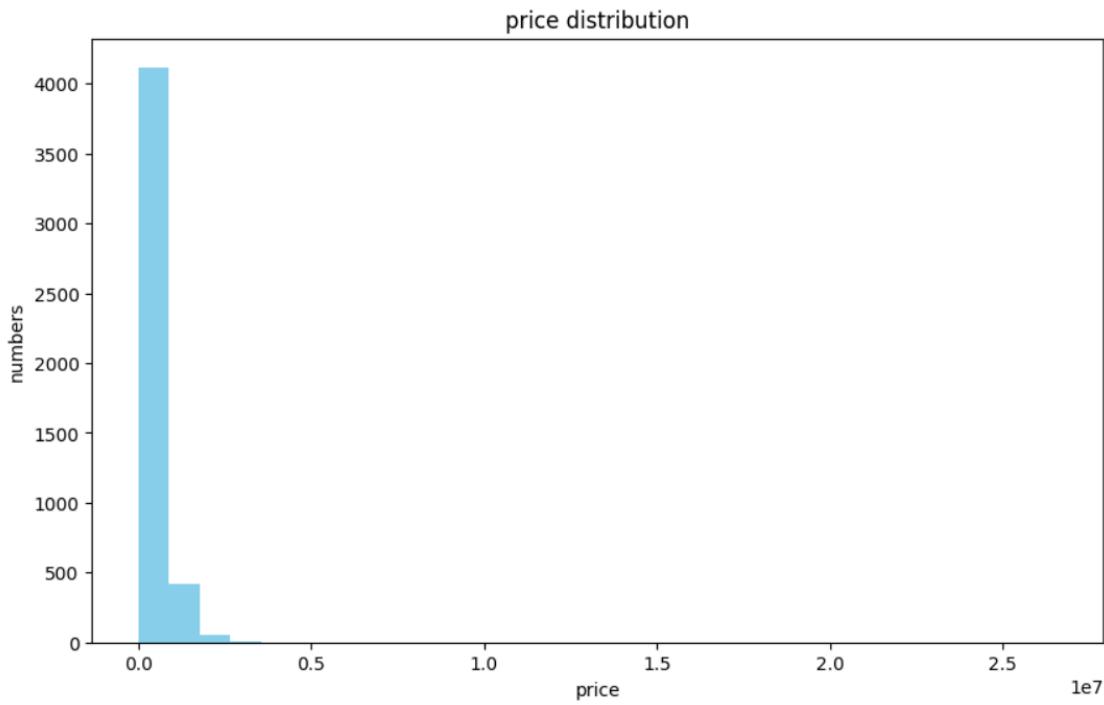
```
# part 3  
plt.figure(figsize=(10, 6))  
plt.hist(data['price'], bins=30, color='skyblue')  
plt.title('price distribution')  
plt.xlabel('price')  
plt.ylabel('numbers')  
plt.show()  
  
plt.figure(figsize=(10, 6))  
plt.scatter(data['price'], data[max_corr_feature_with_price],  
color='coral')  
plt.title(f'price & {max_corr_feature_with_price}')  
plt.xlabel('price')  
plt.ylabel(max_corr_feature_with_price)  
plt.show()
```

این کد دو نمودار را رسم می کند:

۱. نمودار توزیع قیمت در دیتاست:

در این قسمت از تابع plt.hist() که توزیع ستون price را با ۳۰ بین رنگ آبی رسم می کند، استفاده شده است

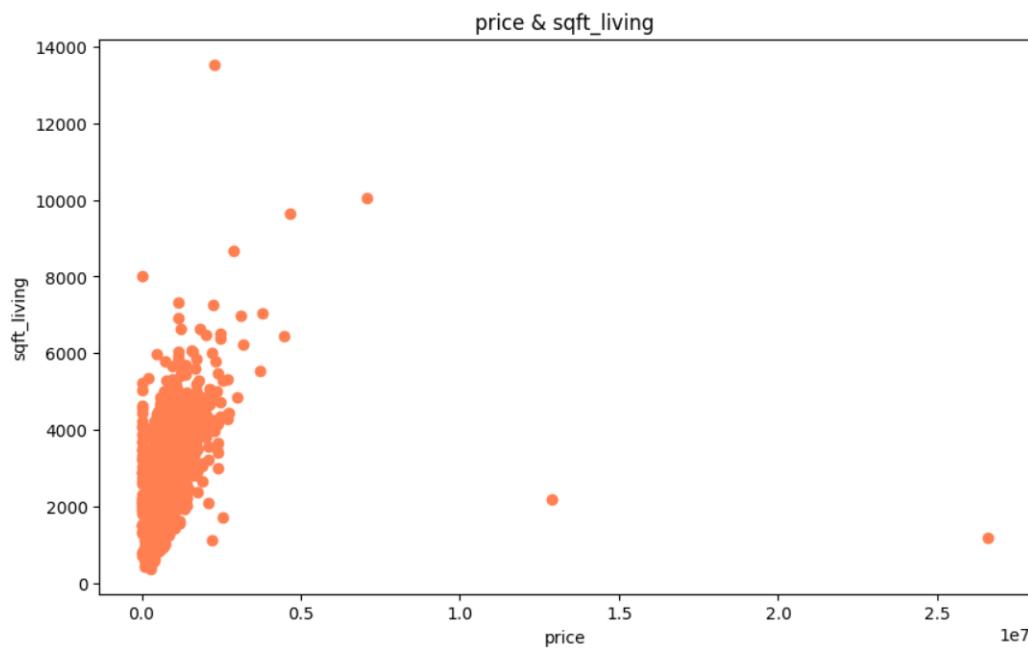
مشاهده خروجی:



۲. نمودار پراکندگی بین ستون قیمت و ویژگی با بیشترین همبستگی با قیمت:

در این قسمت از تابع plt.scatter() پراکندگی نقاط بین دو ستون price و ویژگی با بیشترین همبستگی (که در متغیر max\_corr\_feature\_with\_price ذخیره شده) را که با رنگ قرمز رسم می‌کند، استفاده شده است

مشاهده خروجی:



حال این نمودارها را با کتابخانه Seaborn بررسی می‌کنیم:

```
plt.figure(figsize=(10, 6))
sns.histplot(data['price'], kde=True, color='skyblue')
plt.title('price distribution')
plt.xlabel('price')
plt.ylabel('numbers')
plt.show()

plt.figure(figsize=(10, 6))
sns.scatterplot(x='price', y=max_corr_feature_with_price, data=data,
color='coral')
plt.title(f'price & {max_corr_feature_with_price}')
plt.xlabel('price')
plt.ylabel(max_corr_feature_with_price)
plt.show()
```

همانطور که گفتیم این کد به طور مشابه قبلی دو نمودار را رسم می‌کند با این تفاوت که از توابع داخل کتابخانه استفاده می‌کند:

۱. برای نمودار توزیع تاریخی از تابع sns.histplot() استفاده می‌شود. این تابع به صورت پیشفرض هسته چگالی (Kernel Density Estimation) را هم رسم می‌کند.

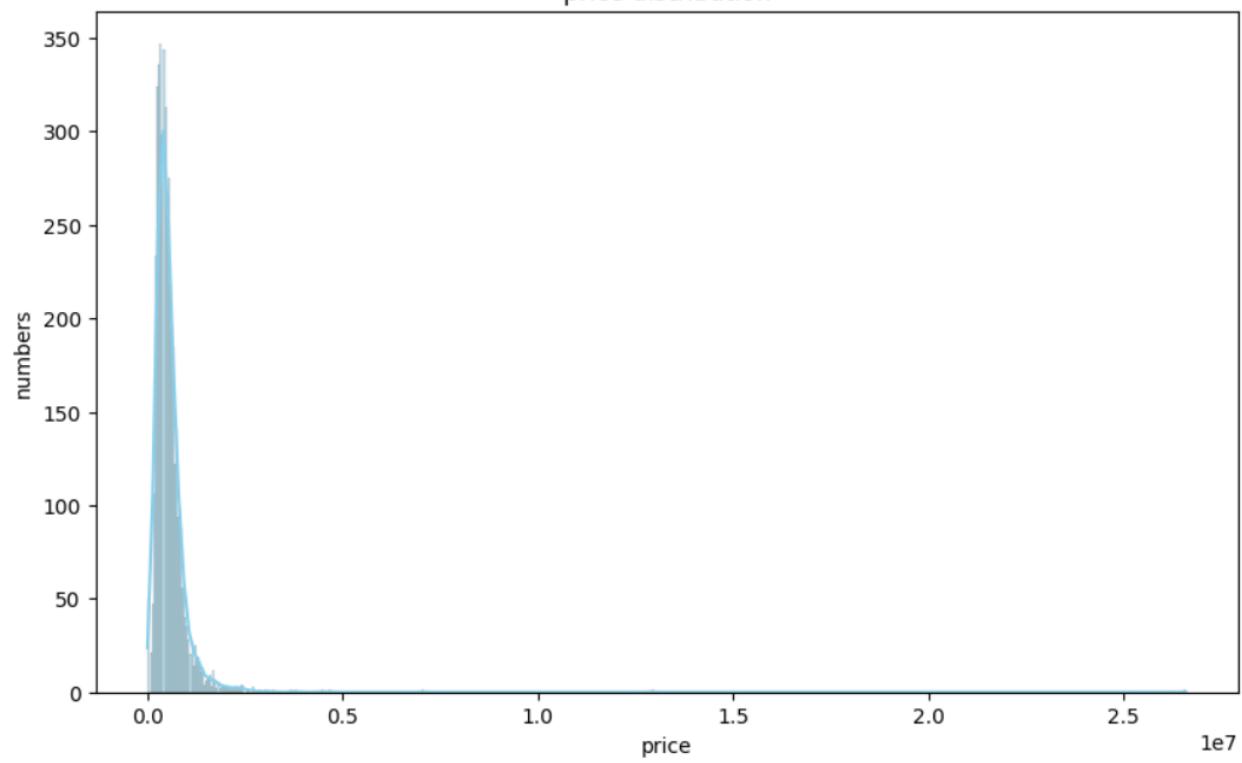
۲. برای نمودار پراکنده‌گی از تابع sns.scatterplot() استفاده می‌شود. در این تابع می‌توان متغیرهای x و y را مستقیماً مشخص کرد.

همانطور که قبلاً هم ذکر شد، استفاده از توابع Seaborn باعث می‌شود نمودارها زیباتر و قابل خواندن‌تر باشند. سایر امکانات مانند تنظیمات رنگ و اندازه هم پیش‌فرض‌تر هستند.

حال خروجی را مشاهده می‌کنیم:

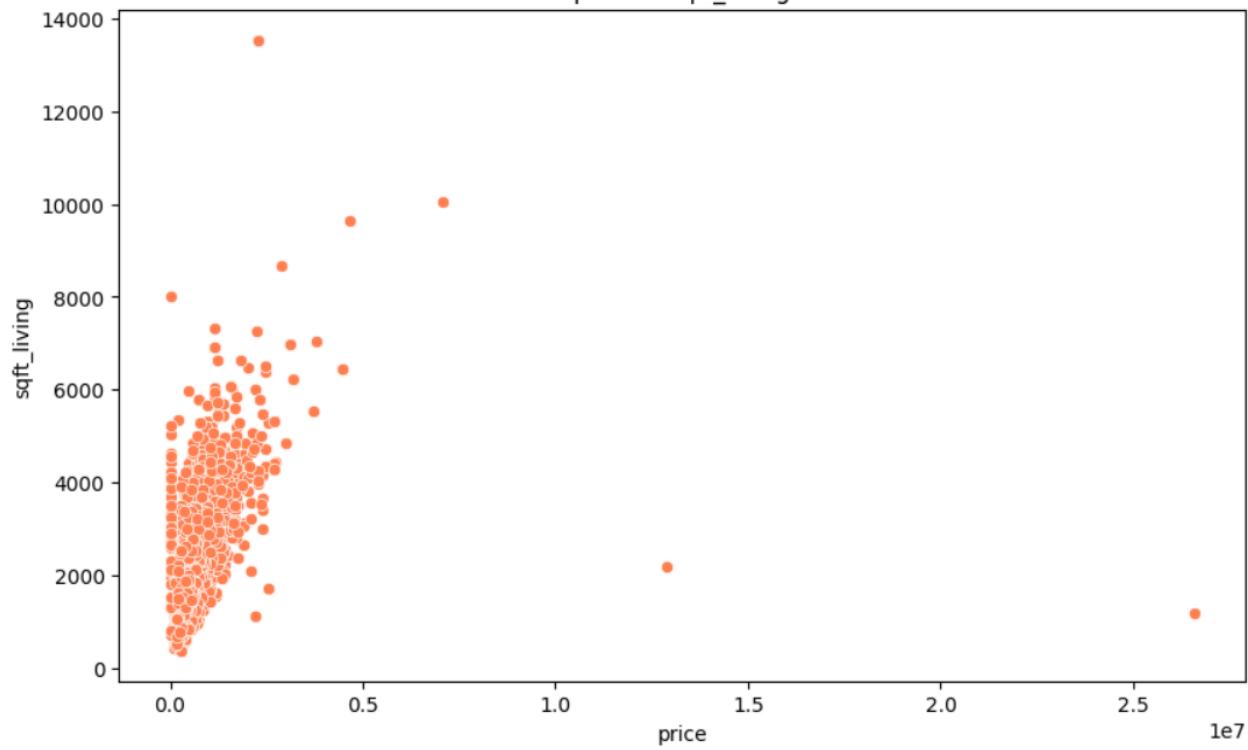
نمودار توزیع قیمت:

price distribution



نمودار قیمت و ویژگی ای که همبستگی زیادی با قیمت دارد:

price & sqft\_living



## ۴-۴ قسمت

ستون Date را به دو ستون ماه و سال تبدیل کنید و این ستون را از دیتافریم حذف کنید.

ابتدا ستون Date را به دو ستون ماه و سال تبدیل می‌کنیم:

```
# part 4
data['date'] = pd.to_datetime(data['date'])
data['month'] = data['date'].dt.month
data['year'] = data['date'].dt.year
```

برای این کار ابتدا تابع pd.to\_datetime() را به فرمت تاریخ-زمان تبدیل می‌کند. سپس دو ستون 'month' و 'year' اضافه می‌شود که مقادیر آنها از داده‌های تاریخ ستون 'date' استخراج می‌شود.

در ادامه ستون Date را از دیتافریم حذف می‌کنیم:

```
data = data.drop('date', axis=1)
```

برای این کار از تابع drop() استفاده می‌کنیم.

در ضمن عبارت axis=1 مشخص می‌کند که ستون‌ها (که در محور ۱ قرار دارند) حذف شوند، نه سطرها.

بنابراین این کد ستون تاریخ را که قبلاً تبدیل به ماه و سال شده بود حذف می‌کند تا از این به بعد فقط با ماه و سال کار شود و نیازی به نگهداری ستون اولیه تاریخ نباشد.

حال با فراخوانی data می‌توان تغییرات را در خروجی مشاهده کرد:

yr_renovated	street	city	statezip	country	month	year
2005	18810 Densmore Ave N	Shoreline	WA 98133	USA	5	2014
0	709 W Blaine St	Seattle	WA 98119	USA	5	2014
0	26206- 26214 143rd Ave SE	Kent	WA 98042	USA	5	2014
0	857 170th PI NE	Bellevue	WA 98008	USA	5	2014
1992	9105 170th Ave NE	Redmond	WA 98052	USA	5	2014
...	...	...	...	...	...	...
1979	501 N 143rd St	Seattle	WA 98133	USA	7	2014
2009	14855 SE 10th Pl	Bellevue	WA 98007	USA	7	2014
0	759 Ilwaco Pl NE	Renton	WA 98059	USA	7	2014

مشاهده می شود که ۲ ستون جدید در سمت راست به داده های ما اضافه شد.

#### ۵-۴

داده ها را با نسبت ۸۰ به ۲۰ درصد به مجموعه های آموزش و آزمون تقسیم کنید و داده های آموزشی و آزمون را با استفاده از MinMaxScaler مقیاس کنید.

ابتدا داده های اضافی یعنی داده هایی که شامل عدد و دیتاست ما نیستند را تشخیص می کنیم و از دیتافریم حذف می کنیم:

```
# part 5
dummy = ['country', 'statezip', 'city', 'street']
data2 = pd.get_dummies(data, columns=dummy, drop_first=True)
data2
```

این کد ستون های موجود در لیست 'dummy' را به صورت 'dummy/indicator' متغیرها در می آورد.

تئضیحات مورد نیاز کد:

تابعی است برای تبدیل ستون‌های کیفی به dummy متفاوتانه است pd.get\_dummies(). مشخص می‌کند که ستون‌های موجود در لیست columns=dummy تبدیل شوند. به معنی حذف کردن یکی از مقادیر دسته‌بندی شده برای هر ستون جهت اجتناب از چندگانگی است.

در نهایت data2 را ایجاد می‌کنیم که دیتا فریم جدیدی است شامل ستون‌های با فرمت dummy به جای ستون‌های اصلی.

حال خروجی data2 را مشاهده می‌کنیم:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	...	street_Indian Trail	street_Nuthatch Trail	street_SE 170th Pl	street_SE 21st Ct	street_Schmitz Park to Alki Trail	street_Shangri-La Way NW
0	3.130000e+05	3.0	1.50	1340	7912	1.5	0	0	3	1340	...	0	0	0	0	0	0
1	2.384000e+06	5.0	2.50	3650	9050	2.0	0	4	5	3370	...	0	0	0	0	0	0
2	3.420000e+05	3.0	2.00	1930	11947	1.0	0	0	4	1930	...	0	0	0	0	0	0
3	4.200000e+05	3.0	2.25	2000	8030	1.0	0	0	4	1000	...	0	0	0	0	0	0
4	5.500000e+05	4.0	2.50	1940	10500	1.0	0	0	4	1140	...	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4595	3.081667e+05	3.0	1.75	1510	6360	1.0	0	0	4	1510	...	0	0	0	0	0	0
4596	5.343333e+05	3.0	2.50	1460	7573	2.0	0	0	3	1460	...	0	0	0	0	0	0
4597	4.169042e+05	3.0	2.50	3010	7014	2.0	0	0	3	3010	...	0	0	0	0	0	0
4598	2.034000e+05	4.0	2.00	2090	6630	1.0	0	0	3	1070	...	0	0	0	0	0	0
4599	2.206000e+05	3.0	2.50	1490	8102	2.0	0	0	4	1490	...	0	0	0	0	0	0

در ادامه داده‌ها را نرم‌الایز می‌کنیم:

```
l1 = LabelEncoder()

for i in data2.columns:
    if data2[i].dtype == 'object':
        data2[i] = l1.fit_transform(data2[i])

data2
```

این کد داده‌های کیفی ستون‌هایی که دارای نوع داده object هستند را به صورت عددی می‌کند.

خروجی data2 را مشاهده می‌کنیم:

سپس X و y هایمان را تعیین می کنیم:

```
X = data2.drop(["price"], axis=1)
y = data2["price"]
X, y
```

گفتیم که این کد داده‌ها را به دو بخش مستقل  $X$  و  $y$  تقسیم می‌کند که  $X$  شامل تمام ستون‌های داده بجز ستون "price" می‌شود و  $y$  تنها شامل ستون "price" می‌شود که به عنوان تابع هدف در نظر گرفته می‌شود. پس این کد داده را برای مدل‌سازی آماده می‌کند.

## مشاهده مقادیر X و y :

```

    street_Schmitz Park to Alki Trail  street_Shangri-La Way NW \
0                           0                   0
1                           0                   0
2                           0                   0
3                           0                   0
4                           0                   0
...
4595                      0                   0
4596                      0                   0
4597                      0                   0
4598                      0                   0
4599                      0                   0

    street_Sunrise Loop Trail  street_Tolt Pipeline Trail \
0                           0                   0
1                           0                   0
2                           0                   0
3                           0                   0
4                           0                   0
...
4595                      0                   0
4596                      0                   0
4597                      0                   0
4598                      0                   0
4599                      0                   0

    street_Trossachs Blvd SE  street_Valley View Trail
0                           0                   0
1                           0                   0
2                           0                   0
3                           0                   0
4                           0                   0
...
4595                      0                   0
4596                      0                   0
4597                      0                   0
4598                      0                   0
4599                      0                   0

```

```

[4600 rows x 4657 columns],
0      3.130000e+05
1      2.384000e+06
2      3.420000e+05
3      4.200000e+05
4      5.500000e+05
...
4595  3.081667e+05
4596  5.343333e+05
4597  4.169042e+05
4598  2.034000e+05
4599  2.206000e+05
Name: price, Length: 4600, dtype: float64)

```

سپس داده‌های آموزش و آزمون را به نسبت ۸۰ به ۲۰ تعیین می‌کنیم:

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
X_train.shape, X_test.shape, y_train.shape, y_test.shape

```

حال ابعاد هر کدام را می‌توان به ترتیب در خروجی مشاهده کرد:

```
((3680, 4657), (920, 4657), (3680,), (920,))
```

سپس در ادامه مقدار بیشینه متغیر پاسخ‌های داده آموزشی پیدا می‌شود که می‌تواند برای تحلیل و ارزیابی الگوریتم مورد استفاده قرار گیرد:

```
max_1 = y_train.values  
maximum = 0  
minimum = min(y_train)  
  
for i in range(X_train.shape[0]):  
    if max_1[i] > maximum:  
        maximum = max_1[i]
```

توضیحات لازم برای کد:

ابتدا مقادیر متغیر `y_train` را به صورت آرایه‌ای در متغیر `max_1` ذخیره می‌کنیم. سپس مقدار اولیه متغیر `maximum` که برای نگهداری مقدار بیشینه تعریف شده است را برابر با صفر قرار می‌دهد. بعد از آن مقدار کمینه متغیر `y_train` را با استفاده از تابع `min` بدست می‌آورد. سپس با یک حلقه `for`، مقادیر آرایه `max_1` را یکی یکی بررسی می‌کند. اگر مقدار جاری بیشتر از مقدار متغیر `maximum` بود، آن مقدار را به عنوان مقدار جدید ذخیره می‌کنیم. در نهایت مقدار بیشینه یافته شده را چاپ می‌کنیم.

سپس با استفاده از `MinMaxScaler` داده‌ها را استاندارد و نرمالایز می‌کنیم:

```
scaler = MinMaxScaler()  
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns = X.columns)  
X_train.head(10)  
print(X_train.max())
```

می‌دانیم `MinMaxScaler` یک ابزار استانداردسازی است که مقادیر را در بازه ۰ تا ۱ نرمالیزه می‌کند. با استفاده از تابع `fit_transform` که مقادیر را استانداردسازی کرده و تبدیل می‌کند، مقادیر `X_train` را استانداردسازی می‌کنیم. در ادامه نتیجه استانداردسازی را در یک دیتا فریم با همان ستون‌های اصلی `X` ذخیره می‌کند. سپس ۱۰ سطر اول دیتا فریم استانداردسازی شده را چاپ می‌کنیم. البته مقدار بیشینه هر ستون را چاپ می‌کنیم که باید ۱ باشد تا نشان دهد که استانداردسازی با موفقیت انجام شده است.

مشاهده خروجی:

```

bedrooms           1.0
bathrooms          1.0
sqft_living        1.0
sqft_lot           1.0
floors             1.0
...
street_Shangri-La Way NW 1.0
street_Sunrise Loop Trail 0.0
street_Tolt Pipeline Trail 0.0
street_Trossachs Blvd SE 1.0
street_Valley View Trail 1.0
Length: 4657, dtype: float64

```

همین کار را برای داده‌های آزمون X نیز انجام می‌دهیم:

```
X_test = pd.DataFrame(scaler.fit_transform(X_test), columns = X.columns)
```

در ادامه این عمل را برای داده‌های y\_train و y\_test انجام می‌دهیم:

```

y_train = pd.DataFrame(y_train)
y_test = pd.DataFrame(y_test)

scaler_2 = MinMaxScaler()

y_train = scaler_2.fit_transform(y_train)
y_test = scaler_2.transform(y_test)

```

## ۶-۴ قسمت

یک مدل Multi-Layer Perceptron (MLP) ساده با ۲ لایه پنهان یا بیشتر بسازید. بخشی از داده‌های آموزش را برای اعتبارسنجی کنار بگذارید و با انتخاب بهینه‌ساز و تابع اتفاف مناسب، مدل را آموزش دهید. نمودارهای اتفاف و R2 Score مربوط به آموزش و اعتبارسنجی را رسم و نتیجه را تحلیل کنید.

در این قسمت ما یک مدل با ۳ لایه پنهان را ساختیم:

```

# part 6
model = Sequential()
model.add(Dense(40, activation= 'relu', input_shape =
(X_train.shape[1],)))
model.add(Dense(20, activation= 'relu'))
model.add(Dense(10, activation= 'relu'))
model.add(Dense(1, activation= 'linear'))

```

```
model.summary()
```

می‌دانیم این کدها برای تعریف مدل MLP با سه لایه مخفی است. ابتدا یک مدل Sequential از کتابخانه Keras تعریف می‌شود. سپس یک لایه پنهان با ۴۰ نورون و فعال‌سازی ReLU اضافه می‌شود. شکل ورودی بر اساس تعداد ستون‌های X\_train تعریف می‌شود. در مرحله بعد و لایه پنهان دوم یک لایه پنهان دیگر با ۲۰ نورون و فعال‌سازی ReLU اضافه می‌شود. سپس یک لایه پنهان دیگر به عنوان لایه پنهان سوم با ۱۰ نورون و فعال‌سازی ReLU اضافه می‌شود. در نهایت یک لایه خروجی با ۱ نورون و فعال‌سازی خطی اضافه می‌شود چون مسئله طبقه‌بندی است.

آخر کار خلاصه مدل با تابع summary چاپ می‌شود تا ساختار آن مشخص شود:

Model: "sequential"		
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 40)	186320
dense_1 (Dense)	(None, 20)	820
dense_2 (Dense)	(None, 10)	210
dense_3 (Dense)	(None, 1)	11
=====		
Total params: 187361 (731.88 KB)		
Trainable params: 187361 (731.88 KB)		
Non-trainable params: 0 (0.00 Byte)		

در ادامه تابع اتلاف و بهینه‌سازمان را برای مدلمان تعریف می‌کنیم:

```
model.compile(optimizer='adam', loss='mse')
history = model.fit(X_train, y_train, validation_split=0.2, epochs=100,
batch_size=10)
```

این کد ابتدا مدل را با استفاده از الگوریتم اپتیمایزر آدام و معیار خطای میانگین مربعات (MSE) کامپایل می‌کند.

سپس تابع fit برای آموزش مدل با داده‌های آموزشی X\_train و y\_train فراخوانی می‌شود. در ادامه ۲۰ درصد از داده‌ها به عنوان داده‌های اعتبارسنجی جدا می‌شوند تا از اضافه شدن حافظه به مدل جلوگیری شود.

در این مدل تعداد اپوک آموزش ۱۰۰ و اندازه باچ ۱۰ در نظر گرفته می‌شود. در نهایت تاریخچه آموزش برای بررسی پیشرفت مدل ذخیره می‌شود.

با این کد مدل به صورت تکراری بر روی داده‌ها آموزش می‌یابد تا خطایش کاهش یابد.

سپس معیار اتلاف و R2 Score را بررسی می‌کنیم:

```
loss = model.evaluate(X_test, y_test)
```

این کد برای ارزیابی مدل آموزش دیده بر روی داده‌های آزمون استفاده می‌شود. تابع evaluate مدل را با داده‌های آزمون X\_test و برچسب‌های واقعی y\_test ارزیابی می‌کند و میزان خطای میانگین مربعات (MSE) را برمی‌گرداند. این مقدار خطای نشان دهنده عملکرد مدل بر روی داده‌های غیر دیده است و می‌تواند برای انتخاب بهترین مدل یا تنظیم پارامترهای مدل استفاده شود. با ارزیابی مدل بر روی داده‌های آزمون، جلوگیری می‌شود که مدل بیش از حد به داده‌های آموزشی سازگار شود و توانایی تعمیم یابی آن ارزیابی می‌شود.

مشاهده مقدار Loss :

```
loss: 0.0193
```

: R2 Score بررسی معیار

```
y_pred = model.predict(X_test)
rscore = r2_score(y_test, y_pred)
```

این کدها برای پیش‌بینی مقادیر برچسب‌های داده‌های آزمون و محاسبه ضریب تعیین (R-squared) استفاده می‌شوند. در قسمت predict() داده‌های آزمون X\_test را به عنوان ورودی می‌گیرد و مقادیر پیش‌بینی شده برای برچسب‌ها را با نام y\_pred برمی‌گرداند. در ادامه r2\_score ضریب تعیین (R-squared) را محاسبه می‌کند. این معیار اندازه‌گیری می‌کند که مدل تا چه حد می‌تواند تغییرات واقعی داده‌ها را توضیح دهد.

در این معیار y\_test برچسب‌های واقعی داده‌های آزمون و y\_pred مقادیر پیش‌بینی شده توسط مدل به عنوان ورودی به این تابع داده می‌شوند.

در نهایت rscore را فراخوانی می‌کنیم:

```
rscore
```

مشاهده خروجی:

```
0.05666247688190307
```

در نهایت نمودارهای مربوطه را رسم می‌کنیم:

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.legend(['Training Loss', 'Validation Loss'])
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```

این کدها برای رسم نمودار تغییرات خطای طول دوره‌های آموزش و آزمون استفاده می‌شوند.

در ادامه به توضیحات مختصر راجع به کدهای بالا می‌پردازیم:

history شامل داده‌های تاریخچه آموزش مدل می‌شود که شامل مقادیر خطای برای داده‌های آموزش و آزمون در هر دوره آموزش است.

plt.plot(history.history['loss']) خطای داده‌های آموزش را رسم می‌کند.

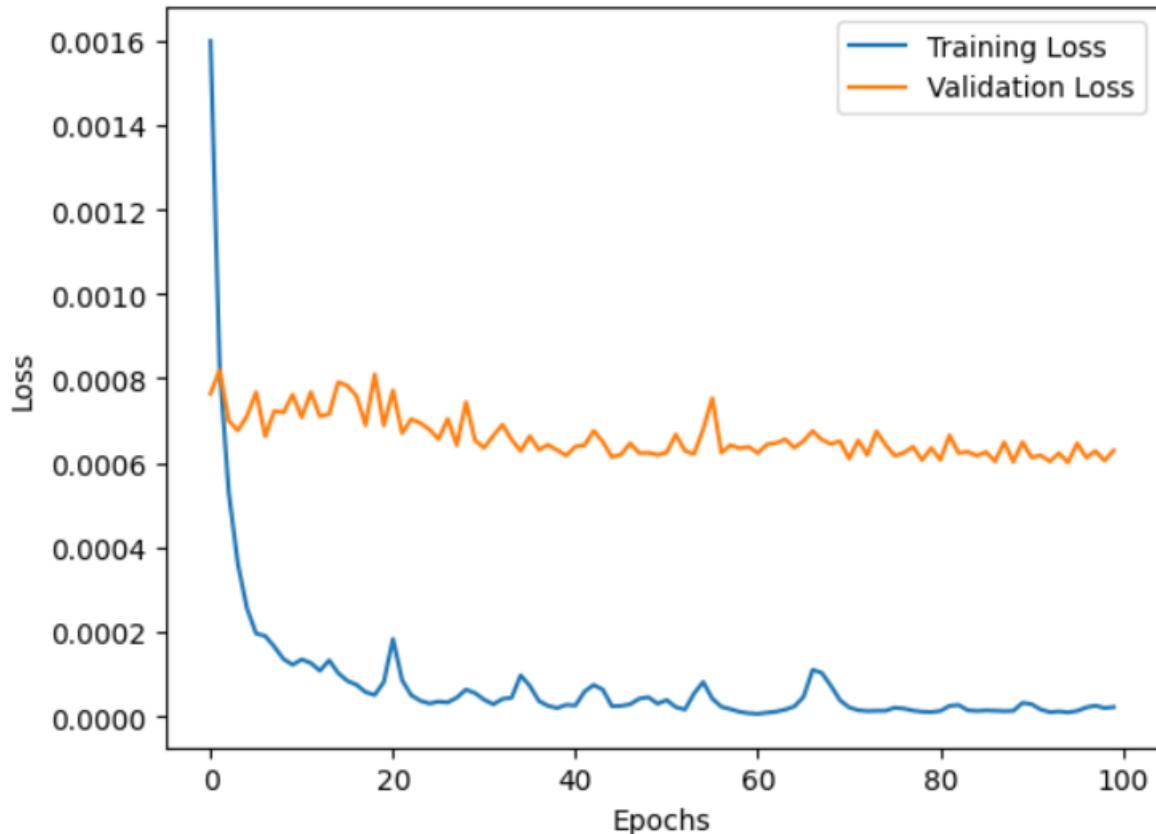
plt.plot(history.history['val\_loss']) خطای داده‌های آزمون را رسم می‌کند.

plt.legend() تایتل برای دو خط رسم شده قرار می‌دهد.

plt.xlabel() و plt.ylabel() عنوان محورها را مشخص می‌کند.

plt.show() نمودار را نمایش می‌دهد تا میزان تطابق مدل با داده‌های آموزش و آزمون در طول دوره‌ها قابل

مشاهده باشد:



مشاهده می شود که در اینجا validation loss می دارای over fitting می باشد که ممکن است در نرمال سازی ما مشکلی باشد.

#### قسمت ۷-۴

فرآیند سوال قبل را با یک بهینه ساز و تابع اتلاف جدید انجام داده و نتایج را مقایسه و تحلیل کنید.

```
# part 7
model_2 = Sequential()
model_2.add(Dense(40, activation= 'relu', input_shape =
(X_train.shape[1],)))
model_2.add(Dense(20, activation= 'relu'))
model_2.add(Dense(10, activation= 'relu'))
model_2.add(Dense(1, activation= 'linear'))
model_2.summary()
```

این مدل با مدل قبلی هیچ فرقی ندارد صرفا اسم آن تغییر یافته است:

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
<hr/>		
dense_4 (Dense)	(None, 40)	186320
dense_5 (Dense)	(None, 20)	820
dense_6 (Dense)	(None, 10)	210
dense_7 (Dense)	(None, 1)	11
<hr/>		
Total params: 187361 (731.88 KB)		
Trainable params: 187361 (731.88 KB)		
Non-trainable params: 0 (0.00 Byte)		

```
model_2.compile(optimizer='sgd', loss='mae')

history = model_2.fit(X_train, y_train, validation_split=0.2, epochs=100,
batch_size=10)
```

در این قسمت بهینه‌ساز را sgd و تابع اتلاف را mae انتخاب کرده‌ایم.

```
loss_2 = model_2.evaluate(X_test, y_test)
loss: 0.0253
```

```
y_pred_2 = model_2.predict(X_test)
rscore_2 = r2_score(y_test, y_pred_2)
```

```
rscore_2
```

: rscore\_2 مشاهده خروجی

```
0.04542789938280567
```

رسم تابع اتلاف:

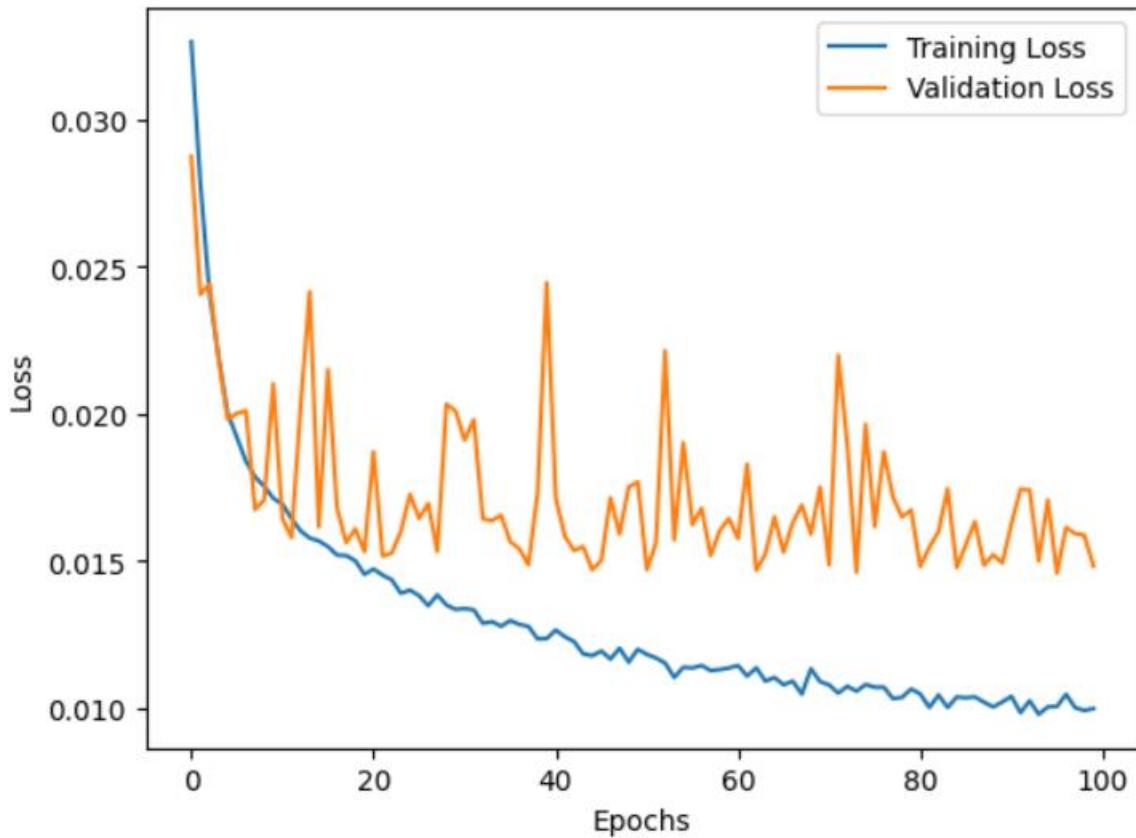
```

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.legend(['Training Loss', 'Validation Loss'])
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()

```

مشاهده خروجی:



مشاهده می شود که در این حالت over fitting بیشتر از حالت قبل است، پس همان بهینه ساز و تابع اتلاف قبلی مناسب تر می باشد.

#### قسمت ۸-۴

پنج داده را به صورت تصادفی از مجموعه ارزیابی انتخاب کرده و قیمت پیشビینی شده را به همراه قیمت واقعی نشان دهید. قیمت پیشビینی شده با قیمت واقعی چقدر تفاوت دارد؟ آیا این عملکرد مناسب است؟ برای بهبود آن چه پیشنهادی دارید؟

ابتدا داده‌های پیش‌بینی شده را با داده‌های واقعی در نمودار اکسٹر مقایسه می‌کنیم:

```
# part 8
y_test_unscaled = scaler_2.inverse_transform(y_test)
y_pred_unscaled = scaler_2.inverse_transform(y_pred)

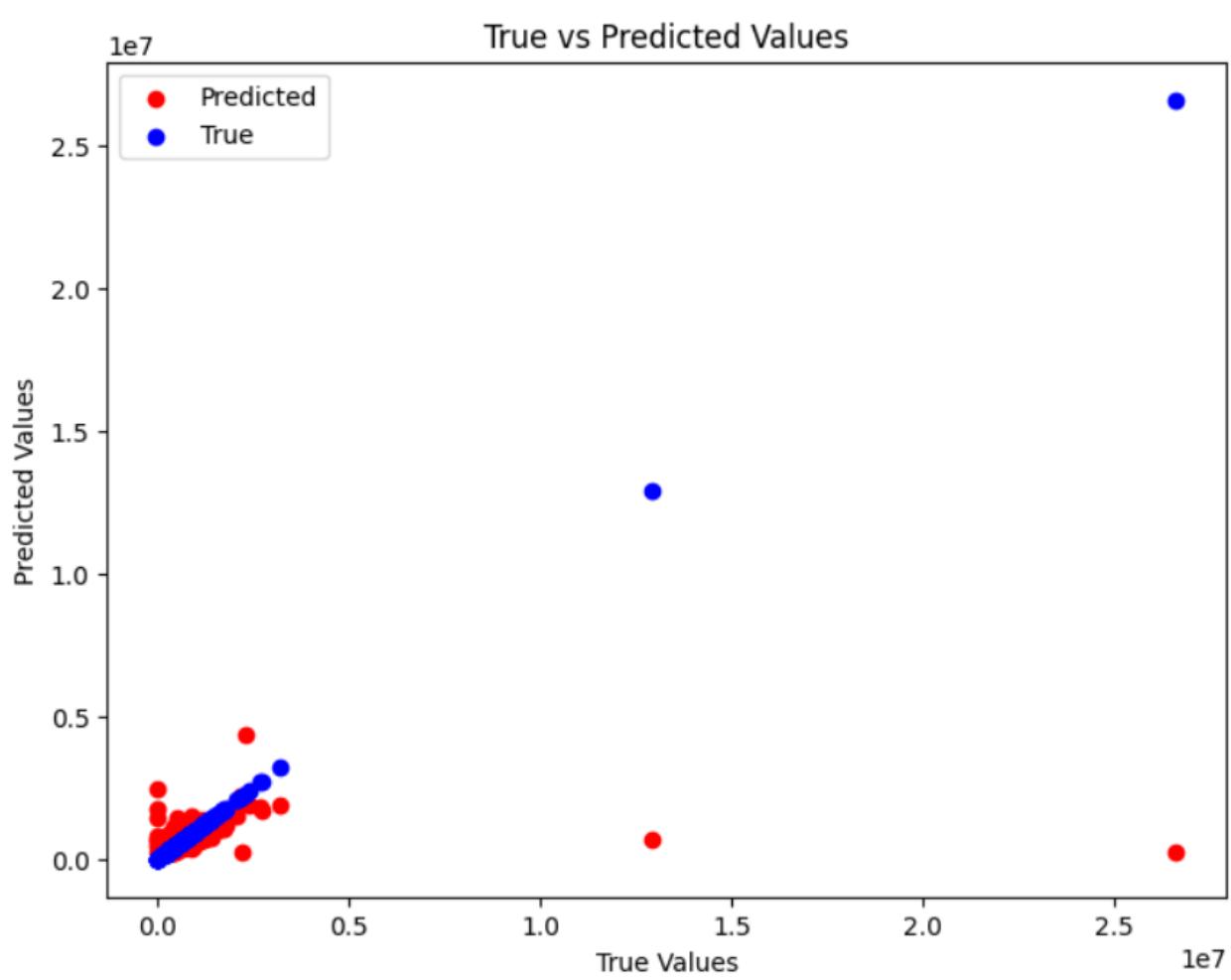
plt.figure(figsize=(8, 6))
plt.scatter(y_test_unscaled, y_pred_unscaled, color='red',
label='Predicted')
plt.scatter(y_test_unscaled, y_test_unscaled, color='blue', label='True')
plt.title('True vs Predicted Values')
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()
```

توضیحات این کد:

scaler\_2 اشاره به ابزار اسکال کردن داده‌ها در مرحله پیش‌پردازش دارد.  
y\_test و y\_pred مقادیر واقعی داده‌های آزمون و پیش‌بینی‌های مدل هستند که در مرحله پیش‌پردازش اسکال شده بودند.

تابعی است که مقادیر را به حالت اولیه قبل از اسکال بر می‌گرداند.  
y\_pred\_unscaled و y\_test\_unscaled مقادیر بدون اسکال شده واقعی و پیش‌بینی شده هستند.  
سپس مشاهده می‌کنیم که دو تا نقطه نمایی با رنگ متفاوت رسم می‌شود تا مقایسه واقعی و پیش‌بینی شده را نشان دهد. در نهایت عناوین و محورها نیز اضافه می‌شود تا نمودار قابل فهم باشد.

مشاهده خروجی:



سپس ۵ داده تصادفی را از داده‌های آزمون انتخاب می‌کنیم:

```
indices = np.random.choice(len(y_test_unscaled), 5, replace=False)

y_test_unscaled_5 = y_test_unscaled[indices]
y test unscaled 5
```

مشاهده خروجی:

```
array([[215000.],
       [375000.],
       [545000.],
       [293000.],
       [380000.]])
```

همین کار را برای داده‌های پیش‌بینی شده انجام می‌دهیم:

```
y_pred_unscaled_5 = y_pred_unscaled[indices]  
y_pred_unscaled_5
```

مشاهده خروجی:

```
array([[265446.1 ],  
       [348227.8 ],  
       [436005.38],  
       [557430.8 ],  
       [451566.16]], dtype=float32)
```

در ادامه میزان اختلاف قیمت‌ها را مشاهده می‌کنیم:

```
errors = np.abs(y_test_unscaled_5 - y_pred_unscaled_5)  
errors
```

مشاهده خروجی:

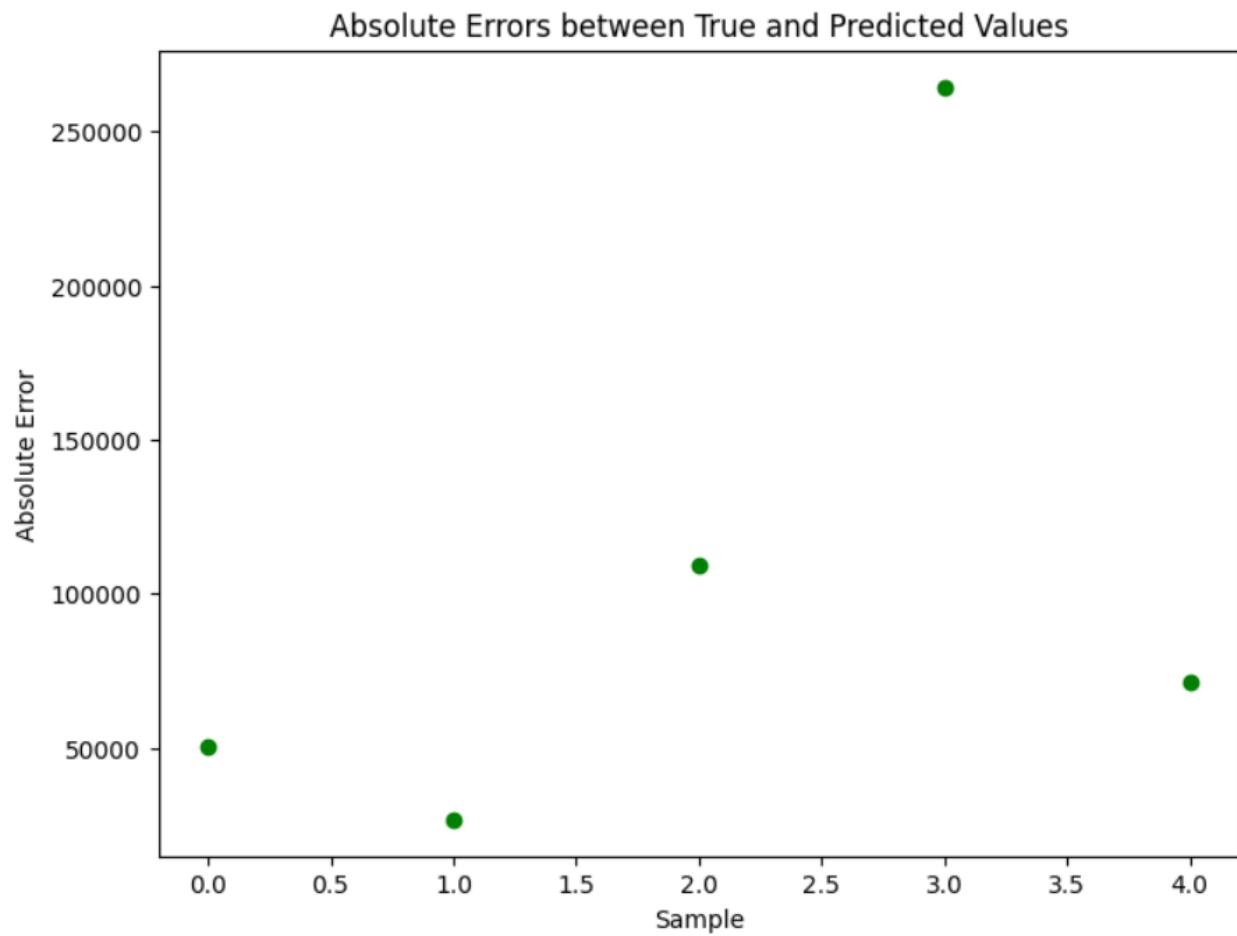
```
array([[ 50446.09375],  
       [ 26772.1875 ],  
       [108994.625 ],  
       [264430.8125 ],  
       [ 71566.15625]])
```

مشاهده می‌شود که در ۵ داده تصادفی انتخاب شده میزان خطأ و اختلاف قیمت، مقادیر قابل توجهی می‌باشند.

برای نشان دادن این موضوع می‌توان از نمودار خطای ۵ داده نیز استفاده کرد:

```
plt.figure(figsize=(8, 6))  
plt.plot(errors, marker='o', linestyle='', color='green')  
plt.title('Absolute Errors between True and Predicted Values')  
plt.xlabel('Sample')  
plt.ylabel('Absolute Error')  
plt.show()
```

مشاهده خروجی:



برای بررسی این نمودار برای همه داده‌ها می‌توان بدین صورت عمل کرد:

```
y_test_unscaled[0]
```

مشاهده خروجی و مقدار آن:

```
array([544000.])
```

حال مقدار پیش‌بینی شده را انتخاب می‌کنیم:

```
y_pred_unscaled[0]
```

مشاهده خروجی:

```
array([551564.94], dtype=float32)
```

حال خط را محاسبه می کنیم:

```
errors = np.abs(y_test_unscaled - y_pred_unscaled)
errors[0]
```

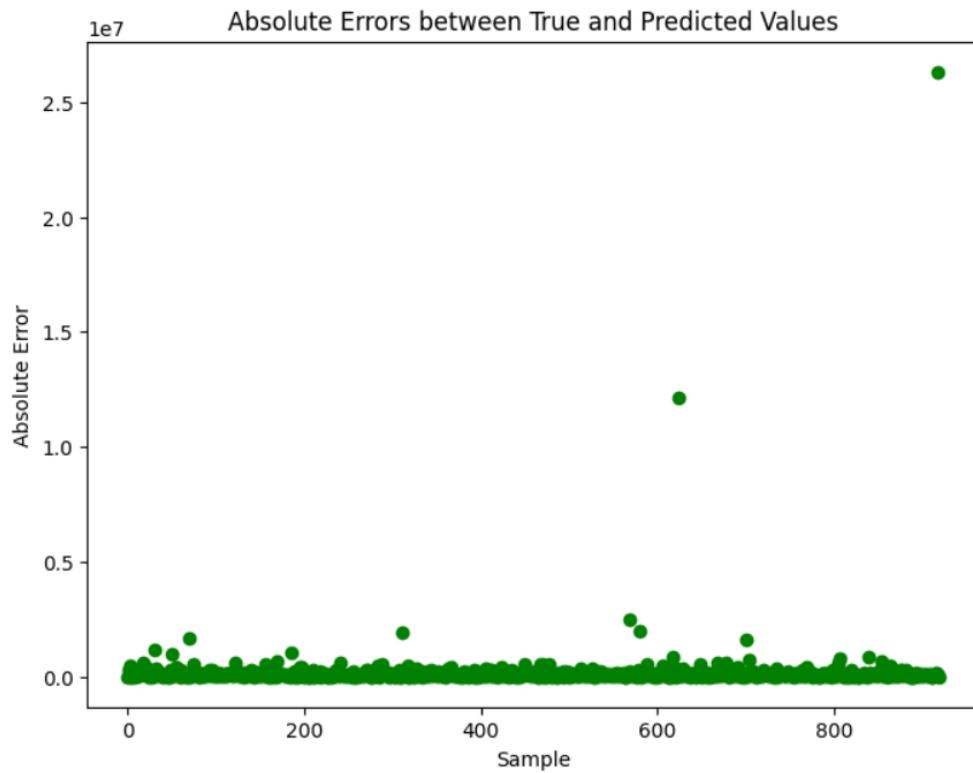
مشاهده خروجی:

```
array([7564.9375])
```

سپس نمودار خط را رسم می کنیم:

```
plt.figure(figsize=(8, 6))
plt.plot(errors, marker='o', linestyle='', color='green')
plt.title('Absolute Errors between True and Predicted Values')
plt.xlabel('Sample')
plt.ylabel('Absolute Error')
plt.show()
```

مشاهده خروجی:



مشاهده می شود که اکثر اختلافات در نزدیکی صفر هستند. پس مدلی که تعریف کردیم قابل قبول می باشد.

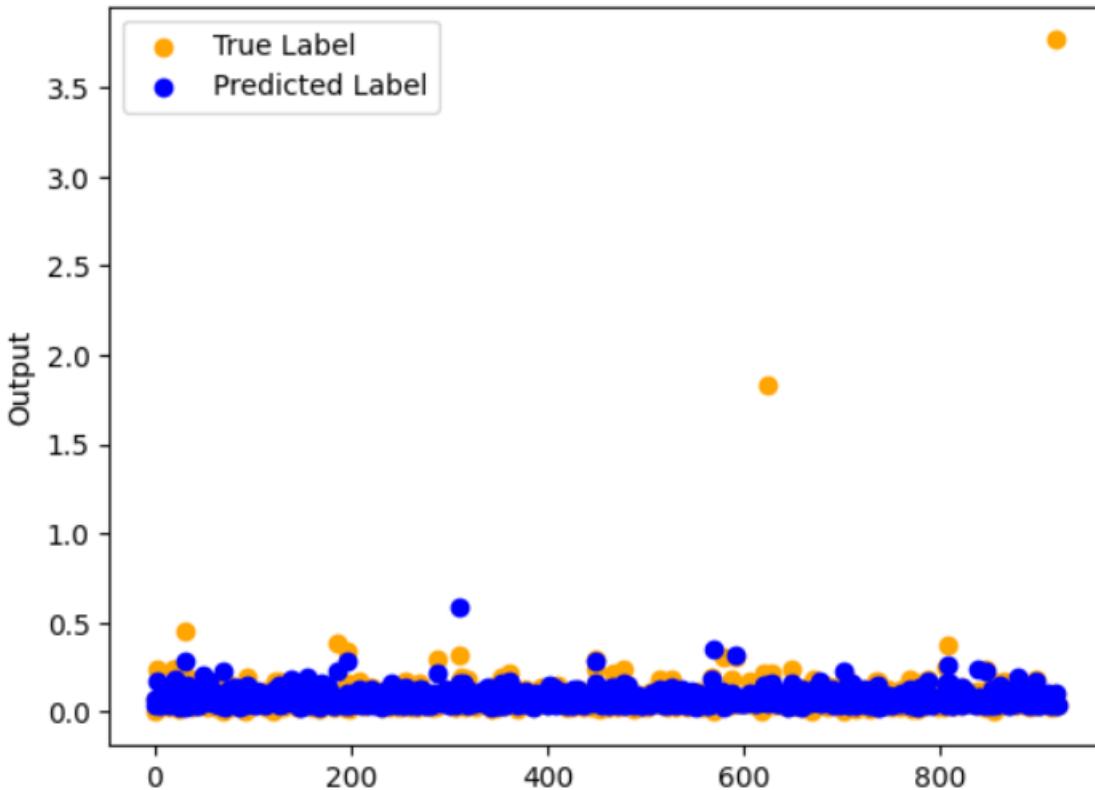
همچنین برای مشاهده داده های اصلی و داده های پیشビینی شده در یک نمودار می توان بدین صورت عمل کرد:

```
# Calculate the range of the output values
a = maximum - minimum

# Convert predicted values back to the original scale
y_pred_true = a * (y_pred)
y_pred_true_1 = y_pred_true + minimum
# Create a scatter plot for true and predicted outputs
plt.scatter(range(len(y_test)), y_test, color="orange") # True labels in orange
plt.scatter(range(len(y_test)), y_pred, color="blue") # Predicted labels in blue

plt.legend(['True Label', 'Predicted Label'])
plt.ylabel("Output")
plt.show()
```

مشاهده خروجی:



در ادامه به مقایسه داده هایمان می پردازیم:

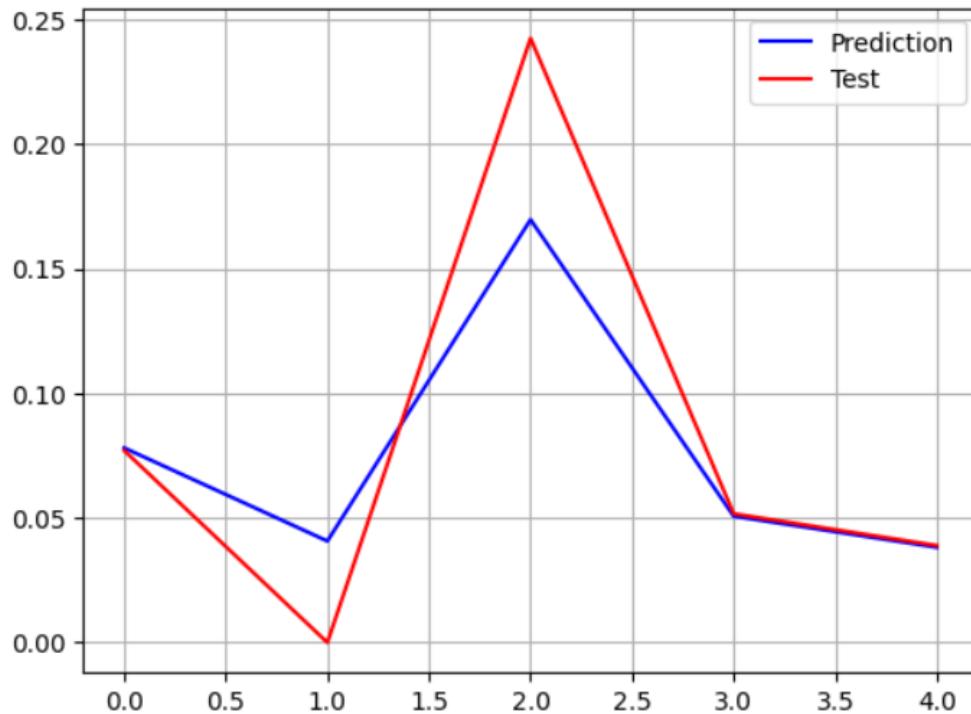
```
random_pred = list()
random_test = list()

for i in range(5):
    j = np.random.randint(0, len(y_pred))
    random_pred.append(y_pred[i])
    random_test.append(y_test[i])

plt.plot(random_pred, 'b', label='Prediction')
plt.plot(random_test, 'r', label='Test')

plt.legend()
plt.grid()
plt.show()
```

مشاهده خروجی:



مشاهده می شود که پیش‌بینی ما، پیش‌بینی خوبی است ولی خیلی دقیق عمل نکرده است (از خطای به دست آمده در بالا نیز می‌توان به این نتیجه رسید). برای بهبود آن می‌توان از بهینه‌سازی‌های متفاوت و یا حتی تابع اتلاف‌های متفاوت استفاده کرد. در ضمن می‌توان در لایه‌های پنهان ایجاد شده توسط مدلمان نیز تغییراتی ایجاد کرد.

## سوال ۵

### قسمت ۱-۵

مجموعه داده Iris را فراخوانی کنید و روش‌های تحلیل داده‌ای که آموخته‌اید را روی آن به کار بینید. داده‌ها را با نسبتی دلخواه و مناسب به مجموعه‌های آموزش و ارزیابی تقسیم کنید.

ابتدا کتابخانه‌های مورد نیاز این قسمت را وارد می‌کنیم:

```
# part 1
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

سپس با استفاده از دستور زیر مجموعه داده Iris را فراخوانی می‌کنیم:

```
iris = load_iris()
```

در ادامه به تحلیل این مجموعه داده می‌پردازیم:

```
iris.data
```

با استفاده از دستور بالا می‌توانیم داده‌ها را مشاهده کنیم.

مشاهده خروجی کد:

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
```

[5.7, 3.8, 1.7, 0.3],  
[5.1, 3.8, 1.5, 0.3],  
[5.4, 3.4, 1.7, 0.2],  
[5.1, 3.7, 1.5, 0.4],  
[4.6, 3.6, 1., 0.2],  
[5.1, 3.3, 1.7, 0.5],  
[4.8, 3.4, 1.9, 0.2],  
[5., 3., 1.6, 0.2],  
[5., 3.4, 1.6, 0.4],  
[5.2, 3.5, 1.5, 0.2],  
[5.2, 3.4, 1.4, 0.2],  
[4.7, 3.2, 1.6, 0.2],  
[4.8, 3.1, 1.6, 0.2],  
[5.4, 3.4, 1.5, 0.4],  
[5.2, 4.1, 1.5, 0.1],  
[5.5, 4.2, 1.4, 0.2],  
[4.9, 3.1, 1.5, 0.2],  
[5., 3.2, 1.2, 0.2],  
[5.5, 3.5, 1.3, 0.2],  
[4.9, 3.6, 1.4, 0.1],  
[4.4, 3., 1.3, 0.2],  
[5.1, 3.4, 1.5, 0.2],  
[5., 3.5, 1.3, 0.3],  
[4.5, 2.3, 1.3, 0.3],  
[4.4, 3.2, 1.3, 0.2],  
[5., 3.5, 1.6, 0.6],  
[5.1, 3.8, 1.9, 0.4],  
[4.8, 3., 1.4, 0.3],  
[5.1, 3.8, 1.6, 0.2],  
[4.6, 3.2, 1.4, 0.2],  
[5.3, 3.7, 1.5, 0.2],  
[5., 3.3, 1.4, 0.2],  
[7., 3.2, 4.7, 1.4],  
[6.4, 3.2, 4.5, 1.5],  
[6.9, 3.1, 4.9, 1.5],  
[5.5, 2.3, 4., 1.3],  
[6.5, 2.8, 4.6, 1.5],  
[5.7, 2.8, 4.5, 1.3],  
[6.3, 3.3, 4.7, 1.6],  
[4.9, 2.4, 3.3, 1.],  
[6.6, 2.9, 4.6, 1.3],  
[5.2, 2.7, 3.9, 1.4],  
[5., 2., 3.5, 1.],  
[5.9, 3., 4.2, 1.5],  
[6., 2.2, 4., 1.],  
[6.1, 2.9, 4.7, 1.4],  
[5.6, 2.9, 3.6, 1.3],  
[6.7, 3.1, 4.4, 1.4],  
[5.6, 3., 4.5, 1.5],  
[5.8, 2.7, 4.1, 1.],  
[6.2, 2.2, 4.5, 1.5],  
[5.6, 2.5, 3.9, 1.1],  
[5.9, 3.2, 4.8, 1.8],

[6.1, 2.8, 4. , 1.3],  
[6.3, 2.5, 4.9, 1.5],  
[6.1, 2.8, 4.7, 1.2],  
[6.4, 2.9, 4.3, 1.3],  
[6.6, 3. , 4.4, 1.4],  
[6.8, 2.8, 4.8, 1.4],  
[6.7, 3. , 5. , 1.7],  
[6. , 2.9, 4.5, 1.5],  
[5.7, 2.6, 3.5, 1. ],  
[5.5, 2.4, 3.8, 1.1],  
[5.5, 2.4, 3.7, 1. ],  
[5.8, 2.7, 3.9, 1.2],  
[6. , 2.7, 5.1, 1.6],  
[5.4, 3. , 4.5, 1.5],  
[6. , 3.4, 4.5, 1.6],  
[6.7, 3.1, 4.7, 1.5],  
[6.3, 2.3, 4.4, 1.3],  
[5.6, 3. , 4.1, 1.3],  
[5.5, 2.5, 4. , 1.3],  
[5.5, 2.6, 4.4, 1.2],  
[6.1, 3. , 4.6, 1.4],  
[5.8, 2.6, 4. , 1.2],  
[5. , 2.3, 3.3, 1. ],  
[5.6, 2.7, 4.2, 1.3],  
[5.7, 3. , 4.2, 1.2],  
[5.7, 2.9, 4.2, 1.3],  
[6.2, 2.9, 4.3, 1.3],  
[5.1, 2.5, 3. , 1.1],  
[5.7, 2.8, 4.1, 1.3],  
[6.3, 3.3, 6. , 2.5],  
[5.8, 2.7, 5.1, 1.9],  
[7.1, 3. , 5.9, 2.1],  
[6.3, 2.9, 5.6, 1.8],  
[6.5, 3. , 5.8, 2.2],  
[7.6, 3. , 6.6, 2.1],  
[4.9, 2.5, 4.5, 1.7],  
[7.3, 2.9, 6.3, 1.8],  
[6.7, 2.5, 5.8, 1.8],  
[7.2, 3.6, 6.1, 2.5],  
[6.5, 3.2, 5.1, 2. ],  
[6.4, 2.7, 5.3, 1.9],  
[6.8, 3. , 5.5, 2.1],  
[5.7, 2.5, 5. , 2. ],  
[5.8, 2.8, 5.1, 2.4],  
[6.4, 3.2, 5.3, 2.3],  
[6.5, 3. , 5.5, 1.8],  
[7.7, 3.8, 6.7, 2.2],  
[7.7, 2.6, 6.9, 2.3],  
[6. , 2.2, 5. , 1.5],  
[6.9, 3.2, 5.7, 2.3],  
[5.6, 2.8, 4.9, 2. ],  
[7.7, 2.8, 6.7, 2. ],  
[6.3, 2.7, 4.9, 1.8],

```
[6.7, 3.3, 5.7, 2.1],  
[7.2, 3.2, 6., 1.8],  
[6.2, 2.8, 4.8, 1.8],  
[6.1, 3., 4.9, 1.8],  
[6.4, 2.8, 5.6, 2.1],  
[7.2, 3., 5.8, 1.6],  
[7.4, 2.8, 6.1, 1.9],  
[7.9, 3.8, 6.4, 2. ],  
[6.4, 2.8, 5.6, 2.2],  
[6.3, 2.8, 5.1, 1.5],  
[6.1, 2.6, 5.6, 1.4],  
[7.7, 3., 6.1, 2.3],  
[6.3, 3.4, 5.6, 2.4],  
[6.4, 3.1, 5.5, 1.8],  
[6., 3., 4.8, 1.8],  
[6.9, 3.1, 5.4, 2.1],  
[6.7, 3.1, 5.6, 2.4],  
[6.9, 3.1, 5.1, 2.3],  
[5.8, 2.7, 5.1, 1.9],  
[6.8, 3.2, 5.9, 2.3],  
[6.7, 3.3, 5.7, 2.5],  
[6.7, 3., 5.2, 2.3],  
[6.3, 2.5, 5., 1.9],  
[6.5, 3., 5.2, 2. ],  
[6.2, 3.4, 5.4, 2.3],  
[5.9, 3., 5.1, 1.8]])
```

هر یک از ستون‌های خروجی بالا در حقیقت همان X‌های ما هستند.

```
iris.target
```

با استفاده از کد بالا می‌توانیم برچسب‌های داده را مشاهده کنیم:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,  
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,  
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

این مقادیر عملاً همان y ما هستند.

```
iris.target_names
```

دستور بالا برچسب‌های کلاس را به ما نشان می‌دهد.

مشاهده خروجی :

```
array(['setosa', 'versicolor', 'virginica'], dtype='|<U10')
```

```
iris.feature_names
```

یا استفاده از کد یالا می توانیم نام هر فیچر را بدانیم:

مشاهده خروجی:

```
['sepal length (cm)',  
 'sepal width (cm)',  
 'petal length (cm)',  
 'petal width (cm)']
```

iris.DESCR

با استفاده از کد بالا می‌توان توضیحات مجموعه داده، مشاهده کرد:

```
.. iris dataset:\n\nIris plants dataset\n-----\n\n**Data Set Characteristics:**\n :Number of Instances: 150 (50 in each of three classes)\n :Number of Attributes: 4 numeric, predictive attributes and the class\n :Attribute Information:\n - sepal length in cm\n - sepal width in cm\n - petal length in cm\n - petal width in cm\n - class:\n - Iris-Setosa\n - Iris-Versicolour\n - Iris-Virginica\n\n :Summary Statistics:\n ======\n Min Max Mean SD Class Correlation\n ======\n\n sepal length: 4.3 7.9 5.84\n 0.83 0.7826\n sepal width: 2.0 4.4 3.05 0.43 -0.4194\n petal length: 1.0\n 6.9 3.76 1.76 0.9490 (high!)\n petal width: 0.1 2.5 1.20 0.76 0.9565\n (high!)\n\n :Missing Attribute Values: None\n :Class Distribution: 33.3% for each of 3 classes.\n :Creator: R.A. Fisher\n :Donor: Michael Marshall\n (MARSHALL%PLU@io.arc.nasa.gov)\n :Date: July, 1988\n\nThe famous Iris database, first used by Sir R.A. Fisher. The dataset is taken\nfrom Fisher's paper. Note that it's the same as in R, but not as in the UCI\nMachine Learning Repository, which has two wrong data points.\n\nThis is perhaps the best known database to be found in the\npattern recognition literature. Fisher's paper is a classic in the field and\nis referenced frequently to this day. (See Duda & Hart, for example.) The\ndataset contains 3 classes of 50 instances each, where each class refers to
```

```

a\n type of iris plant. One class is linearly separable from the other 2;
the\n latter are NOT linearly separable from each other.\n\n.. topic:: References\n - Fisher, R.A. "The use of multiple measurements in taxonomic problems"\n Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n Mathematical Statistics" (John Wiley, NY, 1950).\n - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.\n (Q327.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.\n - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System\n Structure and Classification Rule for Recognition in Partially Exposed\n Environments". IEEE Transactions on Pattern Analysis and Machine\n Intelligence, Vol. PAMI-2, No. 1, 67-71.\n - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule". IEEE Transactions\n on Information Theory, May 1972, 431-433.\n - See also: 1988 MLC Proceedings, 54-64. Cheeseman et al's AUTOCLASS II\n conceptual clustering system finds 3 classes in the data.\n - Many, many more ...

```

حال داده‌ها را به مجموعه آموزش و آزمون تقسیم می‌کنیم. ما این نسبت را مثل سوال‌های قبل ۲۰ به ۸۰ درصد انتخاب می‌کنیم.

برای این کار ابتدا X و y مان را از داده Iris انتخاب می‌کنیم:

```

X, y = load_iris(return_X_y=True)
print(X.shape, y.shape)

```

مشاهده خروجی و ابعاد X و y :

```
(150, 4) (150,)
```

در ادامه داده‌های غیرمتوازن در مجموعه داده‌های Iris را به حالت تعادل می‌رسانیم:

```

import numpy as np
from imblearn.under_sampling import RandomUnderSampler

unique, counts = np.unique(y, return_counts=True)
print("Class counts before balancing:", dict(zip(unique, counts)))

rus = RandomUnderSampler(random_state=42)
X_resampled, y_resampled = rus.fit_resample(X, y)

unique_resampled, counts_resampled = np.unique(y_resampled,
return_counts=True)

```

```
print("Class counts after balancing:", dict(zip(unique_resampled,
counts_resampled)))
print("New balanced dataset shape:", X_resampled.shape, y_resampled.shape)
```

در کد بالا ابتدا تعداد نمونه‌های هر کلاس در مجموعه داده اصلی محاسبه می‌شود تا میزان غیرتوازنی داده‌ها را نشان دهد. سپس از الگوریتم RandomUnderSampler کتابخانه Imblearn برای تعادل داده‌ها استفاده می‌شود. این الگوریتم با انتخاب تصادفی نمونه‌ها از کلاس‌های غالب، تعداد نمونه‌ها را در همه کلاس‌ها برابر می‌کند. در نهایت تعداد نمونه‌های هر کلاس پس از تعادل داده‌ها چاپ می‌شود تا مشخص شود که کلاس‌ها اکنون تعداد برابری از نمونه دارند و داده‌ها تعادل یافته‌اند.

مشاهده خروجی:

```
Class counts before balancing: {0: 50, 1: 50, 2: 50}
Class counts after balancing: {0: 50, 1: 50, 2: 50}
New balanced dataset shape: (150, 4) (150,)
```

در ادامه می‌خواهیم میزان غیرتوازنی داده‌ها را بدانیم. برای این کار ابتدا خروجی unique یا همان مقادیر منحصر به فرد کلاس‌ها را مشاهده می‌کنیم:

```
unique
```

```
array([0, 1, 2])
```

سپس counts یا همان تعداد تکرار هر مقدار منحصر به فرد پیش از تعادل‌سازی در آرایه اصلی را می‌بینیم:

```
counts
```

```
array([50, 50, 50])
```

در ادامه دو لیست و را به یک دیکشنری تبدیل می‌کنیم. هدف از این کار، دسترسی آسان‌تر به تعداد نمونه‌های هر کلاس، ذخیره‌سازی و بازیابی ساده‌تر اطلاعات و همچنین استفاده آسان‌تر در الگوریتم‌ها و توابع دیگر، می‌باشد. برای این کار از دستور `dict(zip(unique, counts))` استفاده می‌کنیم:

```
dict(zip(unique, counts))
```

راجع به کد بالا باید بگوییم که zip تابعی است که دو یا چند لیست را به صورت زوج‌های عناصر ترکیب می‌کند. یعنی (unique[0], counts[0]) را به صورت زوج‌هایی مانند zip(unique, counts) ... ترکیب می‌کند. این زوج‌ها را به عنوان آیتم‌های دیکشنری در نظر می‌گیرد که کلید آن unique و مقدار آن counts است.

مشاهده خروجی:

```
{0: 50, 1: 50, 2: 50}
```

در ادامه X و y جدیدمان را نام‌گذاری می‌کنیم:

```
X = X_resampled  
y = y_resampled  
print(X.shape, y.shape)
```

همانطور که گفتیم این دستورات مقادیر متغیرهای ورودی X و y را با مقادیر متغیرهای resample شده y\_resampled و X\_resampled جایگزین می‌کند و سپس شکل آنها را چاپ می‌کند تا ببینیم اندازه و بعد جدید متغیرها چقدر است. حال خروجی را مشاهده می‌کنیم:

```
(150, 4) (150,)
```

با متعادل شدن X و y حالا می‌توانیم داده‌ها را تقسیم کنیم:

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2)  
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

مشاهده خروجی:

```
((120, 4), (30, 4), (120,), (30,))
```

## ۲-۵ قسمت

با استفاده از روش های آماده پایتون، سه مدل بر مبنای رگرسیون لجستیک، MLP و شبکه های عصبی پایه شعاعی (RBF) را تعریف کرده و روی داده ها آموزش دهید. نتایج روی داده های ارزیابی را حداقل با چهار شاخص و ماتریس درهم ریختگی نشان داده و تحلیل کنید. در انتخاب فراپارامترها آزاد هستید؛ اما لازم است که نتایج را به صورت کامل مقایسه و تحلیل کنید. به دانشجویانی که این سوال را بدون استفاده از کتابخانه ها و مدل های آماده پایتونی انجام دهند، تا ۲۰ درصد نمره امتیازی تعلق خواهد گرفت.

### مدل بر اساس رگرسیون لجستیک:

در روش اول از کتابخانه های آماده سایکیت لرن استفاده می کنیم:

```
# part 2

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming you've trained your model already
model = LogisticRegression(max_iter=1000, random_state=12)
model.fit(x_train, y_train)

# Making predictions on the test set
y_pred = model.predict(x_test)

# Calculating confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)

# Plotting confusion matrix as a heatmap with fitted text
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues',
annot_kws={"size": 12})

# Get the axis to modify layout
plt.gca().set_yticks(len(np.unique(y_test)), 0) # Fix for matplotlib 3.1.1
and 3.1.2
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Save the plot as PNG
plt.tight_layout()
```

```

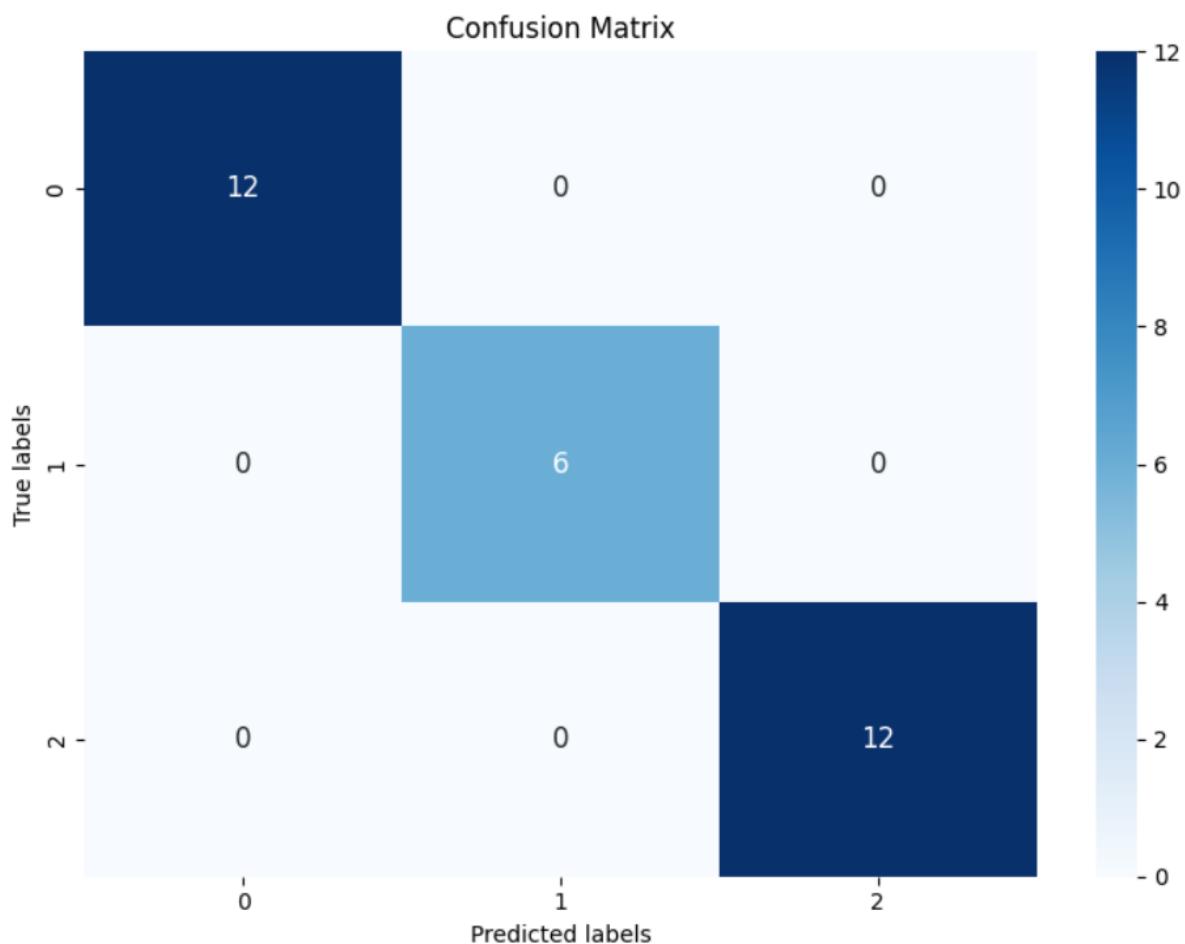
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()

# Printing classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

```

همانطور که گفتیم قرار است در این قسمت از مدل رگرسیون لجستیک برای طبقه‌بندی استفاده شود. ابتدا مدل با داده‌های آموزشی fit می‌شود. سپس پیش‌بینی‌ها بر روی داده‌های آزمون انجام می‌شود. بعد از آن ماتریس درهم‌ریختگی (Confusion Matrix) برای ارزیابی مدل محاسبه می‌شود و سپس به صورت نمودار حرارتی نمایش داده می‌شود. در نهایت گزارش طبقه‌بندی (Classification Report) چاپ می‌شود تا ویژگی‌های مدل و شاخص‌های خواسته شده سوال ارزیابی شود.

مشاهده ماتریس درهم‌ریختگی :



بررسی شاخص‌ها و ویژگی‌های مدل:

Classification Report:					
	precision	recall	f1-score	support	
0	1.00	1.00	1.00	12	
1	1.00	1.00	1.00	6	
2	1.00	1.00	1.00	12	
accuracy			1.00	30	
macro avg	1.00	1.00	1.00	30	
weighted avg	1.00	1.00	1.00	30	

با توجه به گزارش داده شده می‌توان نکات زیر را تحلیل کرد:

precision: دقت یعنی نسبت تعداد آیتم‌های طبقه بندی شده صحیح به کل آیتم‌های طبقه بندی شده برای طبقه. برای طبقه ۰ و ۱ که ۱ است نشان می‌دهد تمام آیتم‌های طبقه بندی شده برای این طبقات صحیح بوده است.

recall: بازیابی یعنی نسبت تعداد آیتم‌های طبقه بندی شده صحیح به تعداد واقعی آیتم‌های مربوط به طبقه. این ویژگی برای تمام طبقات ۱ است که نشان‌دهنده بازیابی بالای مدل است.

f1-score: ترکیبی از precision و recall است. نزدیک بودن به ۱ نشان‌دهنده عملکرد بهتر مدل است.

Accuracy: نشان میدهد تقریباً ۱۰۰ درصد از آیتم‌ها را مدل به درستی طبقه بندی کرده است.

macro avg: میانگین ساده از متريک‌هایي مانند precision, recall, f1-score برای هر کلاس می‌باشد. یعنی بدون توجه به تعداد سمپل‌های هر کلاس میانگین گرفته می‌شود.

weighted avg: میانگین وزن دار از متريک‌ها بر اساس تعداد سمپل‌های هر کلاس است. یعنی کلاس‌های با تعداد سمپل بيشتر وزن بيشتری دارند.

support: تعداد واقعی سمپل‌های موجود در هر کلاس از داده تست است که مدل بر روی آن ارزیابی می‌شود. بنابراین بر اساس این گزارش می‌توان گفت مدل طبقه بندی با دقت و بازیابی بالا عمل کرده و توانسته اکثر داده‌ها را به درستی طبقه بندی نماید.

با دستور زیر می‌توان میزان همبستگی هر طبقه را بر اساس درصد نشان داد:

```
# Assuming you've trained your model already
model = LogisticRegression(max_iter=1000, random_state=12)
model.fit(x_train, y_train)

# Making predictions on the test set
y_pred = model.predict(x_test)

# Calculating confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)

# Calculating percentages for each cell
cf_matrix_percent = cf_matrix.astype('float') / cf_matrix.sum(axis=1)[:, np.newaxis] * 100

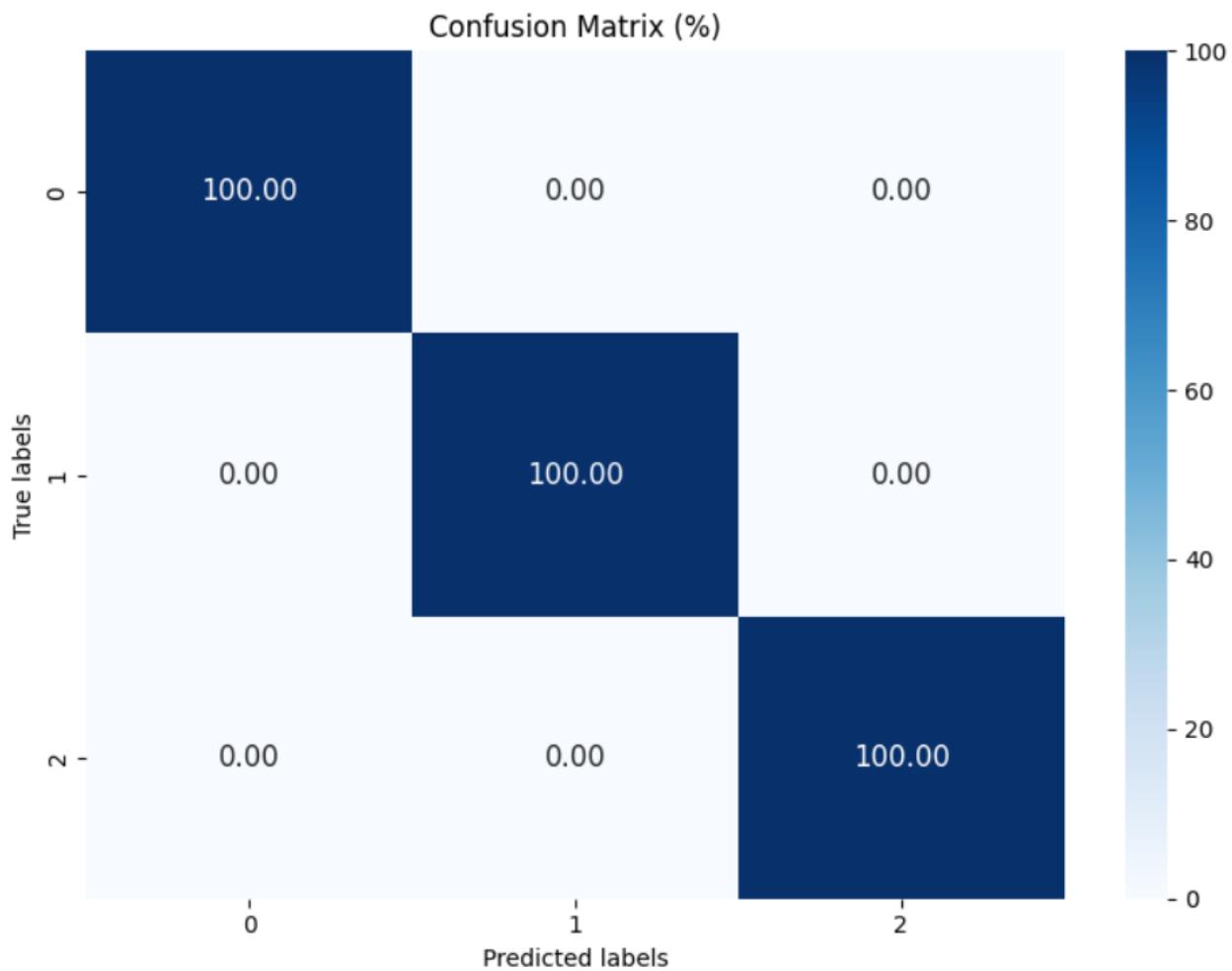
# Plotting confusion matrix as a heatmap with percentages
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix_percent, annot=True, fmt='.2f', cmap='Blues',
            annot_kws={"size": 12})

# Get the axis to modify layout
plt.gca().set_yticks(len(np.unique(y_test)), 0) # Fix for matplotlib 3.1.1
and 3.1.2
plt.title('Confusion Matrix (%)')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Save the plot as PNG
plt.tight_layout()
plt.savefig('confusion_matrix_percentage.png', dpi=300)
plt.show()

# Printing classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

مشاهده خروجی:



در ضمن با استفاده از دستوراتی که آموختیم، می‌توان نمودار خط را نیز رسم کرد:

```
from sklearn.model_selection import validation_curve

# Define the MLPClassifier model
model = LogisticRegression(max_iter=1000, random_state=12)

# Train the model and capture the training/validation scores and losses
train_scores, valid_scores = validation_curve(
    model, x_train, y_train, param_name="max_iter", param_range=[10, 50,
100, 200],
    cv=5, scoring="accuracy", n_jobs=1
)

# Calculate the mean training/validation losses
train_losses = 1 - train_scores.mean(axis=1)
```

```

valid_losses = 1 - valid_scores.mean(axis=1)

# Fit the model with the full training set
model.fit(x_train, y_train)

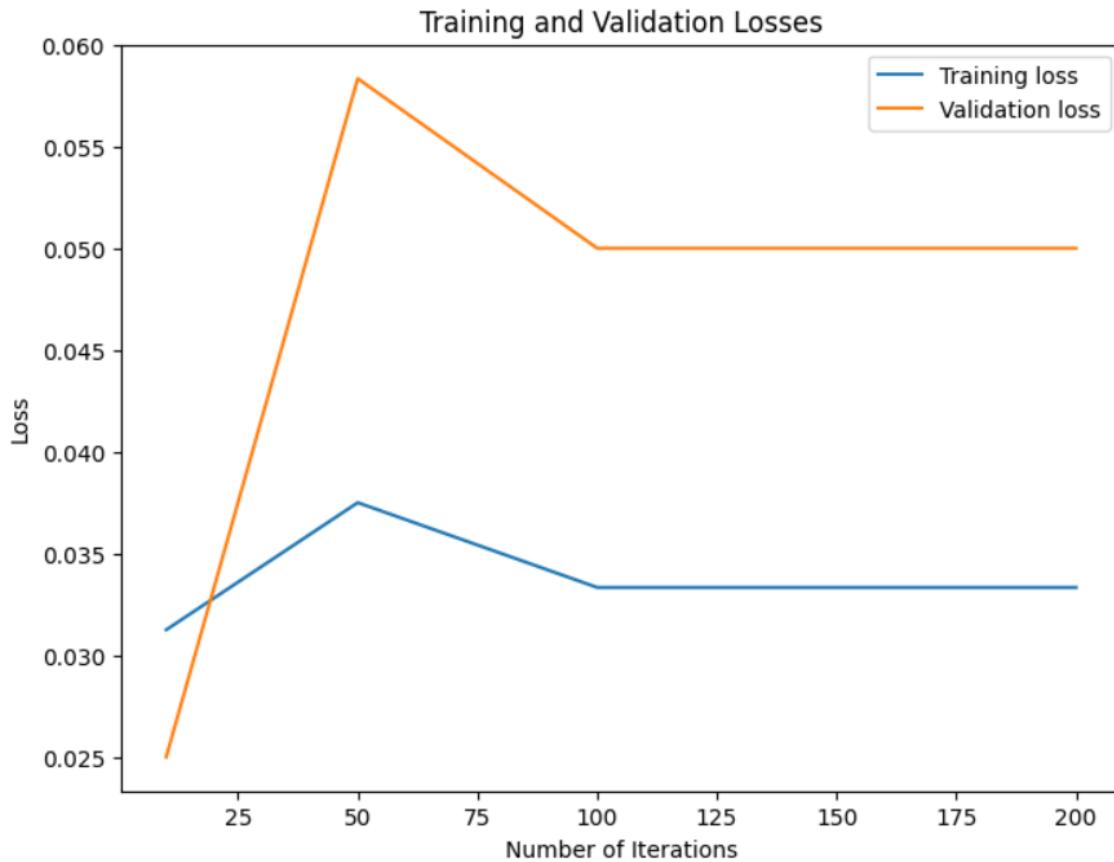
# Calculate the accuracy on the test set
test_accuracy = model.score(x_test, y_test)

# Plotting the training and validation losses
plt.figure(figsize=(8, 6))
plt.plot([10, 50, 100, 200], train_losses, label='Training loss')
plt.plot([10, 50, 100, 200], valid_losses, label='Validation loss')
plt.xlabel('Number of Iterations')
plt.ylabel('Loss')
plt.title('Training and Validation Losses')
plt.legend()
plt.show()

print(f"Test Accuracy: {test_accuracy}")

```

مشاهده خروجی:



حال مدل را بدون استفاده از کتابخانه‌های آماده پایتون تعریف می‌کنیم:

ابتدا کلاس لجستیک رگرسیون را تعریف می‌کنیم:

```
# Define sigmoid function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Logistic regression model class
class LogisticRegression1:

    def __init__(self, learning_rate=0.01, n_iterations=1000):
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations

    def fit(self, X, y):

        # Number of training samples
        n_samples, n_features = X.shape

        # Initialize weights
        self.weights = np.zeros(n_features)

        # Gradient descent
        for i in range(self.n_iterations):

            # Predictions
            y_predicted = self.predict(X)

            # Update weights
            self.weights -= self.learning_rate * (1/n_samples) * np.dot(X.T,
(y_predicted - y))

    def predict(self, X):
        # Sigmoid activation
        return sigmoid(np.dot(X, self.weights))
```

حال به طور دقیق‌تر به بررسی کد می‌پردازیم:

ابتدا تابع sigmoid را که به عنوان تابع فعال ساز در الگوریتم رگرسیون لجستیک استفاده می‌شود، تعریف می‌کنیم که به صورت زیر می‌باشد:

```
return 1 / (1 + np.exp(-x))
```

عملای تابع sigmoid که به صورت فرمول بالا تعریف شده است ورودی  $x$  را به مقداری بین  $0$  و  $1$  تبدیل می‌کند. مرحله بعدی، تعریف یک کلاس به نام LogisticRegression1 که وظیفه پیاده‌سازی مدل رگرسیون لجستیک را دارد.

بررسی این کلاس:

```
LogisticRegression1: def __init__(self, learning_rate=0.01, n_iterations=1000)
```

که مقادیر پیش‌فرض برای نرخ یادگیری و تعداد تکرارها را دریافت می‌کند.

```
: self.learning_rate = learning_rate
```

داده شده است.

```
: self.n_iterations = n_iterations
```

شده است.

```
def fit(self, X, y):
```

تابع fit که وظیفه آموزش مدل رگرسیون لجستیک را دارد. ورودی‌های  $X$  و  $y$  به ترتیب نشان دهنده ویژگی‌ها و برچسب‌های کلاس‌ها هستند.

```
: n_samples, n_features = X.shape
```

```
: self.weights = np.zeros(n_features)
```

دریافت تعداد نمونه‌ها و تعداد ویژگی‌ها از داده‌های ورودی  $X$ . مقداردهی اولیه به وزن‌ها با صفر.

```
: y_predicted = self.predict(X)
```

پیش‌بینی برچسب‌ها با استفاده از تابع predict که در ادامه توضیح داده شده است.

```
: self.weights -= self.learning_rate * (1/n_samples) * np.dot(X.T, (y_predicted - y))
```

وزن‌ها با استفاده از روش کاهش گرادیان.

```
def predict(self, X):
```

سپس تابع predict را تعریف می‌کنیم که این تابع وظیفه پیش‌بینی برچسب‌ها با استفاده از مدل رگرسیون لجستیک را دارد.

در ادامه مدلمان را تعریف می‌کنیم:

```
# Train the model
model = LogisticRegression1()
model.fit(x_train, y_train)

# Making predictions on the test set
y_pred_1 = model.predict(x_test)

# Calculating confusion matrix
cf_matrix = np.zeros((2,2))

y_test_1 = np.round(y_test).astype(int)
y_pred_1 = np.round(y_pred_1).astype(int)

classes = np.unique(y_test_1)
n_classes = len(classes)

cf_matrix = np.zeros((n_classes, n_classes))

for i in range(len(y_test)):
    cf_matrix[y_test_1[i], y_pred_1[i]] += 1

# Plotting confusion matrix as a heatmap with fitted text
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues',
            annot_kws={"size": 12})
sns.heatmap(cf_matrix, annot=True, fmt='.2f', cmap='Blues',
            annot_kws={"size": 12})

# Get the axis to modify layout
plt.gca().set_yticks(len(np.unique(y_test_1)), 0)

plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Save the plot as PNG
plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()

# Printing classification report
print("Classification Report:")
```

```

tp = 0
fp = 0
fn = 0
tn = 0
for i in range(len(y_test)):
    if y_test[i]==1 and y_pred_1[i]==1:
        tp += 1
    elif y_test[i]==1 and y_pred_1[i]==0:
        fn += 1
    elif y_test[i]==0 and y_pred_1[i]==1:
        fp += 1
    elif y_test[i]==0 and y_pred_1[i]==0:
        tn += 1

print("Precision: ", tp/(tp+fp))
print("Recall: ", tp/(tp+fn))
print("Accuracy: ", (tp+tn)/(tp+fp+fn+tn))

```

حال به بررسی این کد می‌پردازیم:

۱. `model = LogisticRegression1()`: یک شی از کلاس `LogisticRegression1` ایجاد می‌شود.
۲. `model.fit(x_train, y_train)`: مدل رگرسیون لجستیک آموزش داده می‌شود با استفاده از داده‌های آموزش `y_train` و برچسب‌های آموزش `x_train`
۳. `y_pred_1 = model.predict(x_test)`: برچسب‌های پیش‌بینی شده برای داده‌های تست با استفاده از مدل رگرسیون لجستیک محاسبه می‌شود.
۴. `y_test = np.round(y_test).astype(int)`: برچسب‌های واقعی برای داده‌های تست گرد می‌شوند و به صورت عدد صحیح ذخیره می‌شوند.
۵. `y_pred_1 = np.round(y_pred_1).astype(int)`: برچسب‌های پیش‌بینی شده برای داده‌های تست گرد می‌شوند و به صورت عدد صحیح ذخیره می‌شوند.
۶. `cf_matrix = np.zeros((n_classes, n_classes))`: یک ماتریس صفر برای محاسبه ماتریس درهم‌ریختگی ایجاد می‌شود.

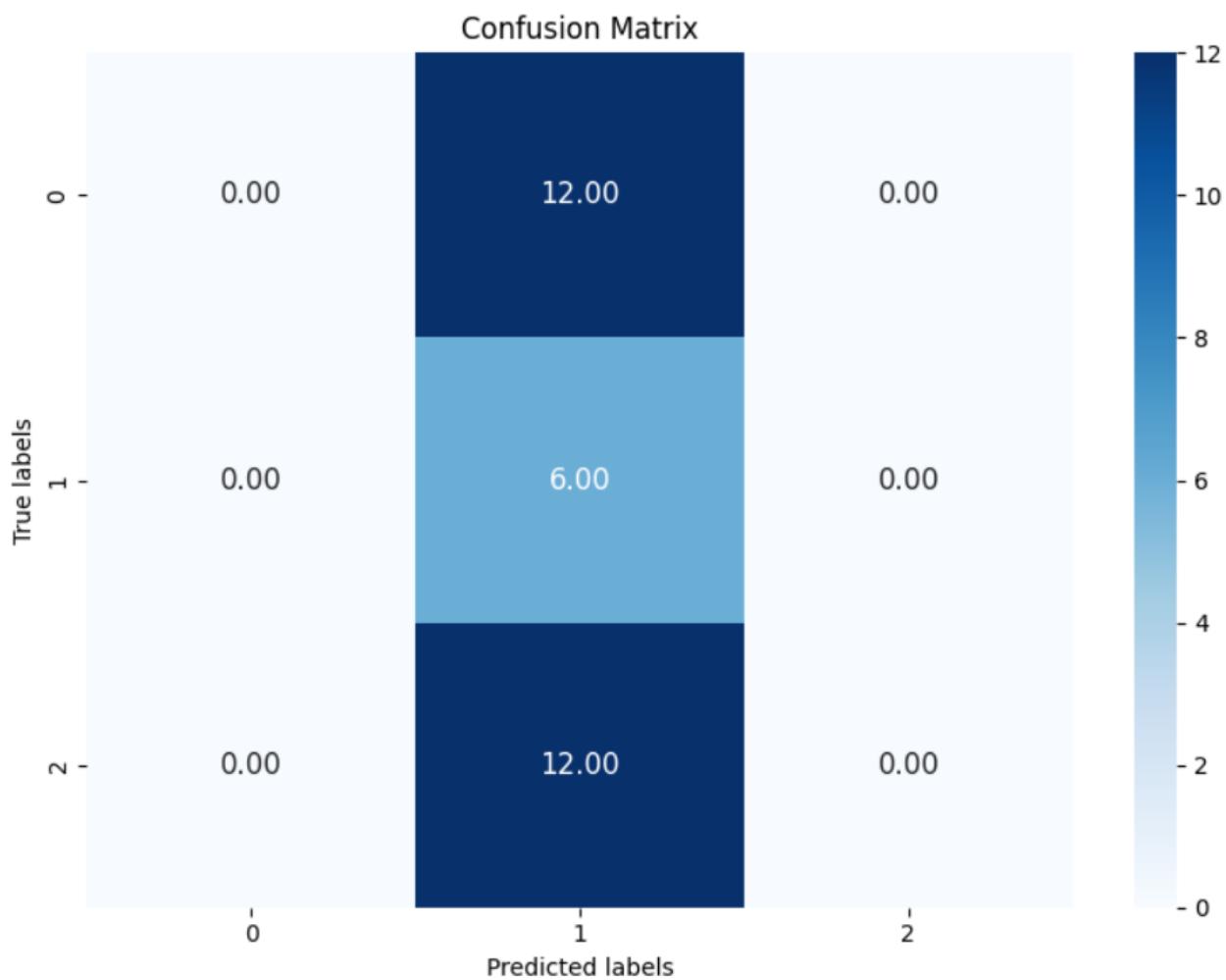
`for i in range(len(y_test)):` .

```
cf_matrix[y_test[i], y_pred_1[i]] += 1`
```

با استفاده از این حلقه ماتریس درهم ریختگی با استفاده از برچسب‌های واقعی و پیش‌بینی شده محاسبه می‌شود.

سپس در ادامه ماتریس درهم ریختگی به صورت heatmap رسم می‌شود و محور y برای نمایش ماتریس درهم ریختگی تغییر می‌کند. در ادامه گزارش دقت، بهره و دقت پیش‌بینی چاپ می‌شود. در این بخش، مقادیر true negative، false negative و false positive، true positive به صورت دستی محاسبه شده‌اند.

مشاهده ماتریس درهم ریختگی :



بررسی شاخص‌ها و ویژگی‌های مدل:

```

Classification Report:
Precision: 0.3333333333333333
Recall: 1.0
Accuracy: 0.3333333333333333

```

با توجه به گزارش دیده شده در خروجی می‌توان گفت که دقت و صحت ما  $33\%$  بوده و نسبت به حالت قبل (مدلی که با استفاده از کتابخانه‌های آماده پایتون طراحی کردیم) پایین‌تر است. پس مدل قبلی مناسب‌تر می‌باشد.

## مدل بر اساس MLP

ابتدا با کتابخانه‌های آماده پایتون مدلمان را تعریف می‌کنیم:

```

from sklearn.neural_network import MLPClassifier
model = MLPClassifier(activation='relu', hidden_layer_sizes=(100,),
                      batch_size='auto', learning_rate='constant',
                      learning_rate_init=0.001,
                      power_t=0.5, max_iter=200, shuffle=True,
                      random_state=None, tol=0.0001, verbose=False,
                      warm_start=False, momentum=0.9,
                      nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1,
                      beta_1=0.9, beta_2=0.999, epsilon=1e-08,
                      n_iter_no_change=10, max_fun=15000,
                      solver='adam')

```

می‌دانیم MLPClassifer کلاس مورد نیاز برای ایجاد یک مدل MLP است. برای این مدل تابع فعال‌سازی `relu` برای لایه‌های پنهان انتخاب شده است که تابع خطی مقطعی را فعال می‌کند. ابتدا یک لایه مخفی با ۱۰۰ گره تعریف شده است. سپس اندازه دسته‌بندی به صورت خودکار تنظیم می‌شود. در این مدل نرخ یادگیری ثابت است و مقدار اولیه آن  $1 \times 10^{-4}$  تنظیم شده است. در ضمن حداکثر تعداد تکرار ۲۰۰ برای آموزش تعریف شده است. در ضمن مشاهده می‌شود که داده‌ها در هر دوره تصادفی می‌شوند. در نهایت برای این مدل الگوریتم بهینه‌سازی Adam مورد استفاده قرار گرفته است.

در ادامه می‌توان نمونه‌های مختلفی برای مدل را تعریف کنیم:

```

model = MLPClassifier(hidden_layer_sizes=(10, 5, 2), random_state=12,
                      verbose = True)
model.fit(x_train, y_train)
model.score(x_test, y_test)

```

این کد شامل سه بخش است:

```
model = MLPClassifier(hidden_layer_sizes=(10, 5, 2), random_state=12, verbose = True) .۱
```

در این قسمت یک شی از کلاس `MLPClassifier` با پارامترهای زیر ایجاد می‌شود:

- تعداد نورون‌های هر لایه مخفی مدل (در اینجا ۳ لایه مخفی با تعداد نورون‌های ۱۰، ۵ و ۲) تعريف شده است.

برای تعیین `seed` در تولید اعداد تصادفی استفاده می‌شود.

برای چاپ پیام‌هایی در هر مرحله از آموزش مدل، این پارامتر را `True` تعیین می‌کنیم.

```
model.fit(x_train, y_train) .۲
```

مدل `MLP` با استفاده از داده‌های آموزش `x_train` و برچسب‌های آموزش `y_train` آموزش داده می‌شود.

```
model.score(x_test, y_test) .۳
```

دقت مدل روی داده‌های تست با استفاده از تابع `score` محاسبه می‌شود.  
دقت مدل برابر است با تعداد پیش‌بینی‌های صحیح (true negative و true positive) تقسیم بر تعداد کل پیش‌بینی‌ها.

مشاهده خروجی:

```
Iteration 1, loss = 2.17862615
Iteration 2, loss = 2.14965671
Iteration 3, loss = 2.12097675
Iteration 4, loss = 2.09258634
Iteration 5, loss = 2.06445899
Iteration 6, loss = 2.03662363
Iteration 7, loss = 2.00909822
Iteration 8, loss = 1.98187180
Iteration 9, loss = 1.95508424
Iteration 10, loss = 1.92863122
Iteration 11, loss = 1.90259781
Iteration 12, loss = 1.87755781
Iteration 13, loss = 1.85251781
Iteration 14, loss = 1.82747781
Iteration 15, loss = 1.80243781
Iteration 16, loss = 1.77739781
Iteration 17, loss = 1.75235781
Iteration 18, loss = 1.72731781
Iteration 19, loss = 1.70227781
Iteration 20, loss = 1.67723781
Iteration 21, loss = 1.65219781
Iteration 22, loss = 1.62715781
Iteration 23, loss = 1.60211781
Iteration 24, loss = 1.57707781
Iteration 25, loss = 1.55203781
Iteration 26, loss = 1.52709781
Iteration 27, loss = 1.50205781
Iteration 28, loss = 1.47701781
Iteration 29, loss = 1.45207781
Iteration 30, loss = 1.42703781
Iteration 31, loss = 1.40209781
Iteration 32, loss = 1.37705781
Iteration 33, loss = 1.35211781
Iteration 34, loss = 1.32717781
Iteration 35, loss = 1.30223781
Iteration 36, loss = 1.27729781
Iteration 37, loss = 1.25235781
Iteration 38, loss = 1.22741781
Iteration 39, loss = 1.20247781
Iteration 40, loss = 1.17753781
Iteration 41, loss = 1.15259781
Iteration 42, loss = 1.12765781
Iteration 43, loss = 1.10271781
Iteration 44, loss = 1.07777781
Iteration 45, loss = 1.05283781
Iteration 46, loss = 1.02789781
Iteration 47, loss = 1.00295781
Iteration 48, loss = 9.77701781
Iteration 49, loss = 9.55207781
Iteration 50, loss = 9.32713781
Iteration 51, loss = 9.10219781
Iteration 52, loss = 8.87725781
Iteration 53, loss = 8.65231781
Iteration 54, loss = 8.42737781
Iteration 55, loss = 8.20243781
Iteration 56, loss = 7.97749781
Iteration 57, loss = 7.75255781
Iteration 58, loss = 7.52761781
Iteration 59, loss = 7.30267781
Iteration 60, loss = 7.07773781
Iteration 61, loss = 6.85279781
Iteration 62, loss = 6.62785781
Iteration 63, loss = 6.40291781
Iteration 64, loss = 6.17797781
Iteration 65, loss = 5.95203781
Iteration 66, loss = 5.72709781
Iteration 67, loss = 5.50215781
Iteration 68, loss = 5.27721781
Iteration 69, loss = 5.05227781
Iteration 70, loss = 4.82733781
Iteration 71, loss = 4.60239781
Iteration 72, loss = 4.37745781
Iteration 73, loss = 4.15251781
Iteration 74, loss = 3.92757781
Iteration 75, loss = 3.70263781
Iteration 76, loss = 3.47769781
Iteration 77, loss = 3.25275781
Iteration 78, loss = 3.02781781
Iteration 79, loss = 2.80287781
Iteration 80, loss = 2.57793781
Iteration 81, loss = 2.35299781
Iteration 82, loss = 2.12705781
Iteration 83, loss = 1.90211781
Iteration 84, loss = 1.67717781
Iteration 85, loss = 1.45223781
Iteration 86, loss = 1.22729781
Iteration 87, loss = 1.00235781
Iteration 88, loss = 7.77741781
Iteration 89, loss = 5.55247781
Iteration 90, loss = 3.32753781
Iteration 91, loss = 1.10259781
Iteration 92, loss = -0.87765781
Iteration 93, loss = -2.10271781
Iteration 94, loss = -3.32777781
Iteration 95, loss = -4.55283781
Iteration 96, loss = -5.77789781
Iteration 97, loss = -7.00295781
Iteration 98, loss = -8.22701781
Iteration 99, loss = -9.45207781
Iteration 100, loss = -10.67713781
Iteration 101, loss = -11.90219781
Iteration 102, loss = -13.12725781
Iteration 103, loss = -14.35231781
Iteration 104, loss = -15.57737781
Iteration 105, loss = -16.80243781
Iteration 106, loss = -18.02749781
Iteration 107, loss = -19.25255781
Iteration 108, loss = -20.47761781
Iteration 109, loss = -21.70267781
Iteration 110, loss = -22.92773781
Iteration 111, loss = -24.15279781
Iteration 112, loss = -25.37785781
Iteration 113, loss = -26.60291781
Iteration 114, loss = -27.82797781
Iteration 115, loss = -29.05203781
Iteration 116, loss = -30.27709781
Iteration 117, loss = -31.50215781
Iteration 118, loss = -32.72721781
Iteration 119, loss = -33.95227781
Iteration 120, loss = -35.17733781
Iteration 121, loss = -36.40239781
Iteration 122, loss = -37.62745781
Iteration 123, loss = -38.85251781
Iteration 124, loss = -40.07757781
Iteration 125, loss = -41.30263781
Iteration 126, loss = -42.52769781
Iteration 127, loss = -43.75275781
Iteration 128, loss = -44.97781781
Iteration 129, loss = -46.20287781
Iteration 130, loss = -47.42793781
Iteration 131, loss = -48.65299781
Iteration 132, loss = -49.87705781
Iteration 133, loss = -51.10211781
Iteration 134, loss = -52.32717781
Iteration 135, loss = -53.55223781
Iteration 136, loss = -54.77729781
Iteration 137, loss = -56.00235781
Iteration 138, loss = -57.22741781
Iteration 139, loss = -58.45247781
Iteration 140, loss = -59.67753781
Iteration 141, loss = -60.90259781
Iteration 142, loss = -62.12765781
Iteration 143, loss = -63.35271781
Iteration 144, loss = -64.57777781
Iteration 145, loss = -65.80283781
Iteration 146, loss = -67.02789781
Iteration 147, loss = -68.25295781
Iteration 148, loss = -69.47701781
Iteration 149, loss = -70.70207781
Iteration 150, loss = -71.92713781
Iteration 151, loss = -73.15219781
Iteration 152, loss = -74.37725781
Iteration 153, loss = -75.60231781
Iteration 154, loss = -76.82737781
Iteration 155, loss = -78.05243781
Iteration 156, loss = -79.27749781
Iteration 157, loss = -80.50255781
Iteration 158, loss = -81.72761781
Iteration 159, loss = -82.95267781
Iteration 160, loss = -84.17773781
Iteration 161, loss = -85.40279781
Iteration 162, loss = -86.62785781
Iteration 163, loss = -87.85291781
Iteration 164, loss = -89.07797781
Iteration 165, loss = -90.30203781
Iteration 166, loss = -91.52709781
Iteration 167, loss = -92.75215781
Iteration 168, loss = -93.97721781
Iteration 169, loss = -95.20227781
Iteration 170, loss = -96.42733781
Iteration 171, loss = -97.65239781
Iteration 172, loss = -98.87745781
Iteration 173, loss = -99.10251781
Iteration 174, loss = -100.32757781
Iteration 175, loss = -101.55263781
Iteration 176, loss = -102.77769781
Iteration 177, loss = -104.00275781
Iteration 178, loss = -105.22781781
Iteration 179, loss = -106.45287781
Iteration 180, loss = -107.67793781
Iteration 181, loss = -108.90299781
Iteration 182, loss = -109.12705781
Iteration 183, loss = -109.35211781
Iteration 184, loss = -109.57717781
Iteration 185, loss = -109.80223781
Iteration 186, loss = -109.10229781
Iteration 187, loss = -109.32735781
Iteration 188, loss = -109.55241781
Iteration 189, loss = -109.77747781
Iteration 190, loss = -109.10253781
Iteration 191, loss = -109.32759781
Iteration 192, loss = -109.55265781
Iteration 193, loss = -109.77771781
Iteration 194, loss = -109.10277781
Iteration 195, loss = -109.32783781
Iteration 196, loss = -109.55289781
Iteration 197, loss = -109.77795781
Iteration 198, loss = -109.10201781
Iteration 199, loss = -109.32707781
Iteration 200, loss = -109.55213781
```

مقدار دقت:

0.4

مشاهده می‌شود که دقت پایین است، پس مدلمان را تغییر می‌دهیم تا دقتش زیاد شود. در این مدل از ۹۰ لایه پنهان استفاده شده است.

```
model = MLPClassifier(hidden_layer_sizes=(90), random_state=12)
model.fit(x_test, y_test)
model.score(x_test, y_test)
```

مشاهده دقت:

0.9333333333333333

این دقت بسیار مناسب می‌باشد و می‌توانیم ادامه سوال را با این مدل حل کنیم.

همانند قسمت قبل و با استفاده از کدهای آن می‌توانیم ماتریس همبستگی را رسم کنیم:

```
# Making predictions on the test set
y_pred = model.predict(x_test)

# Calculating confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)

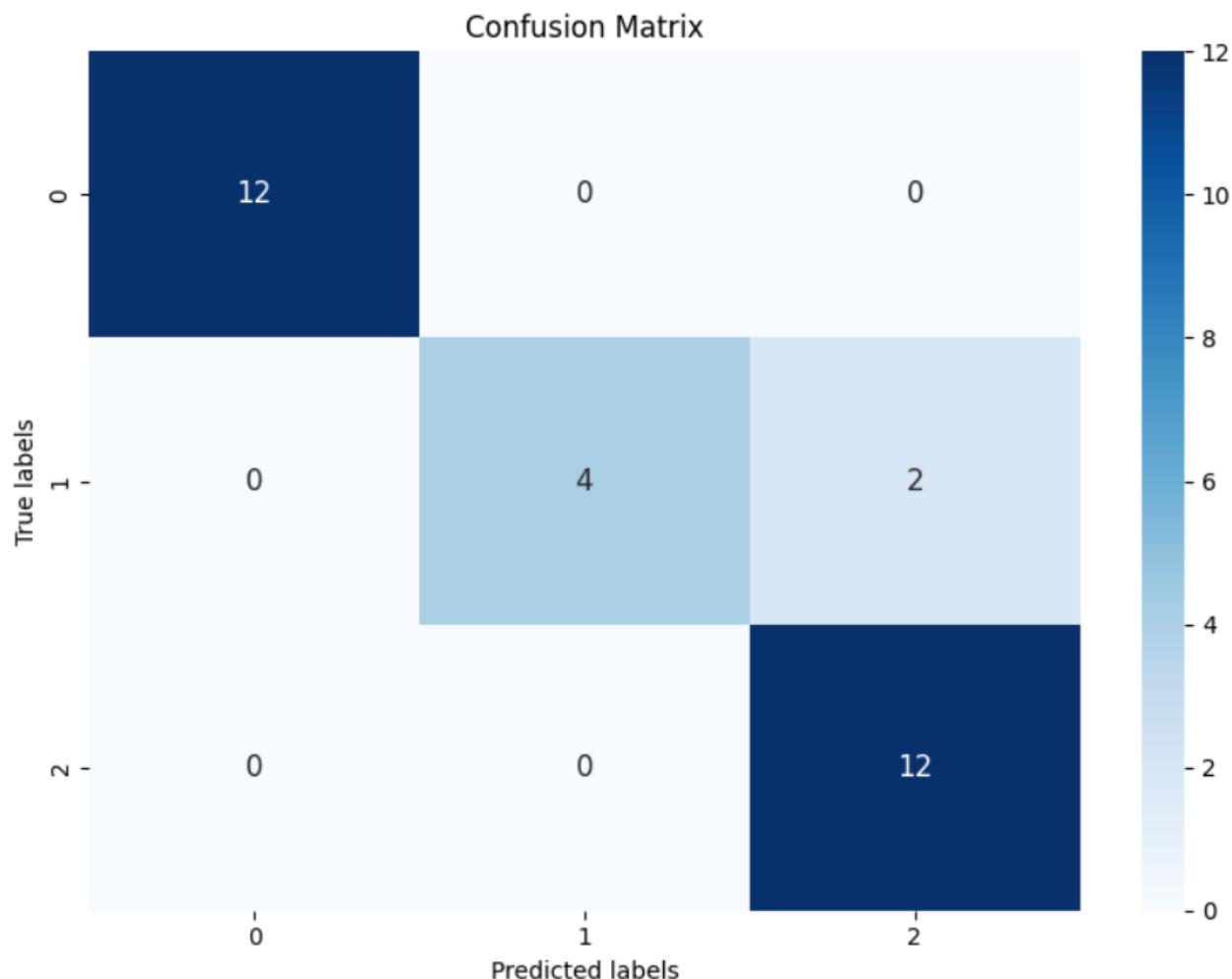
# Plotting confusion matrix as a heatmap with fitted text
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues',
annot_kws={"size": 12})

# Get the axis to modify layout
plt.gca().set_yticks(len(np.unique(y_test)), 0) # Fix for matplotlib 3.1.1
and 3.1.2
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Save the plot as PNG
plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()
```

```
# Printing classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

مشاهده ماتریس درهم ریختگی و بررسی ویژگی های مدل :



Classification Report:				
	precision	recall	f1-score	support
0	1.00	1.00	1.00	12
1	1.00	0.67	0.80	6
2	0.86	1.00	0.92	12
accuracy			0.93	30
macro avg	0.95	0.89	0.91	30
weighted avg	0.94	0.93	0.93	30

با توجه به گزارش داده شده می توان نکات زیر را تحلیل کرد:

precision: برای طبقه ۰ و ۱ است نشان می دهد تمام آیتم های طبقه بندی شده برای این طبقات صحیح بوده است.

recall: این ویژگی برای طبقات ۰ و ۲، ۱ است که نشان دهنده بازیابی نسبتا بالای مدل است.

f1-score: در این مدل نزدیک به ۱ است و مناسب می باشد.

Accuracy: نشان میدهد تقریبا ۹۳ درصد از آیتم ها را مدل به درستی طبقه بندی کرده است.

همانند قسمت قبل می توان میزان همبستگی هر طبقه را بر اساس درصد نشان داد:

```
# Calculating confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)

# Calculating percentages for each cell
cf_matrix_percent = cf_matrix.astype('float') / cf_matrix.sum(axis=1)[:, np.newaxis] * 100

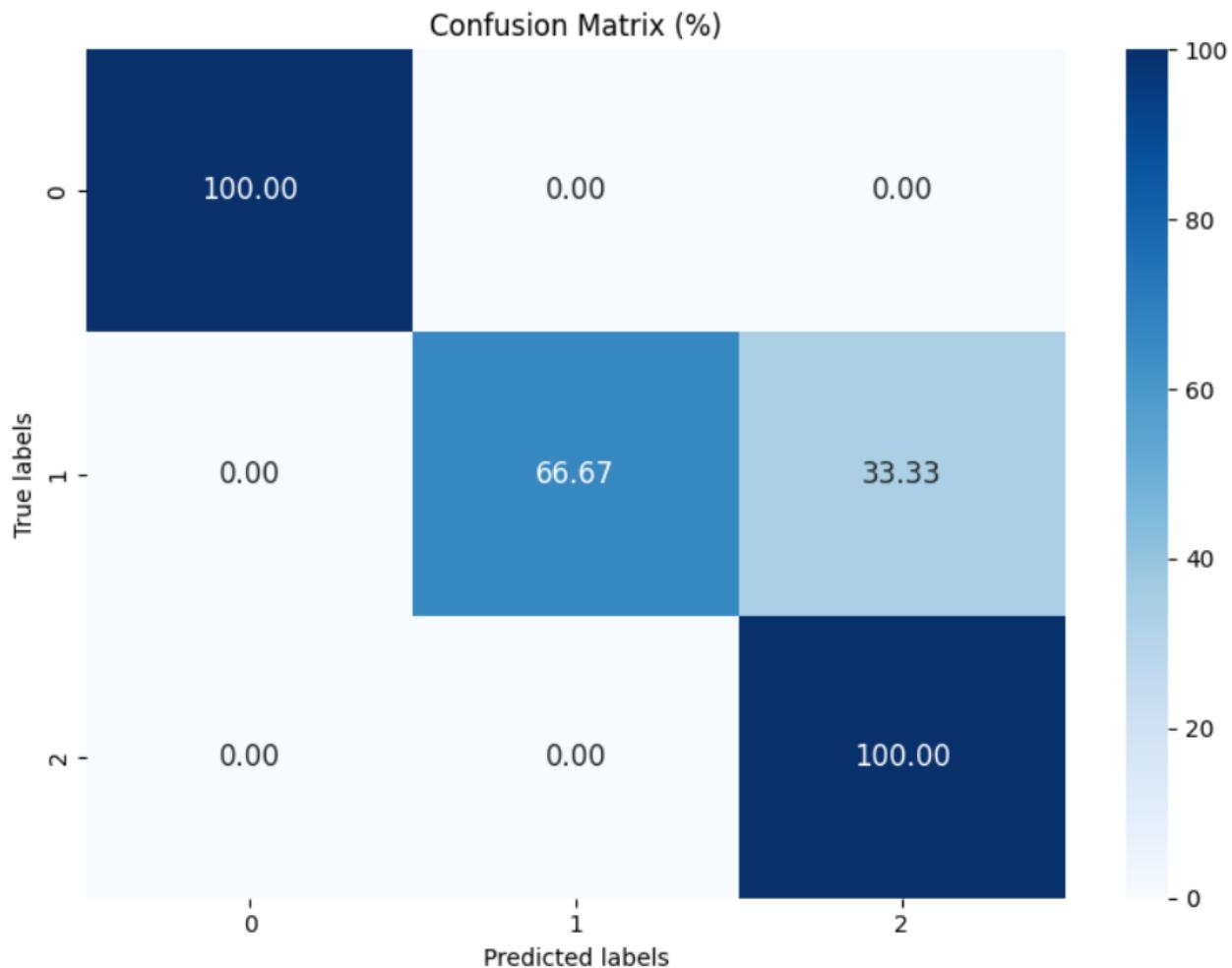
# Plotting confusion matrix as a heatmap with percentages
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix_percent, annot=True, fmt='.2f', cmap='Blues',
annot_kws={"size": 12})

# Get the axis to modify layout
plt.gca().set_yticks(len(np.unique(y_test)), 0) # Fix for matplotlib 3.1.1
and 3.1.2
plt.title('Confusion Matrix (%)')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Save the plot as PNG
plt.tight_layout()
plt.savefig('confusion_matrix_percentage.png', dpi=300)
plt.show()

# Printing classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

مشاهده خروجی:



بهتر است نمودار خط را نیز بررسی کنیم:

```
from sklearn.model_selection import validation_curve

# Train the model and capture the training/validation scores and losses
train_scores, valid_scores = validation_curve(
    model, x_train, y_train, param_name="max_iter", param_range=[10, 50,
100, 200],
    cv=5, scoring="accuracy", n_jobs=1
)

# Calculate the mean training/validation losses
train_losses = 1 - train_scores.mean(axis=1)
valid_losses = 1 - valid_scores.mean(axis=1)
```

```

# Fit the model with the full training set
model.fit(x_train, y_train)

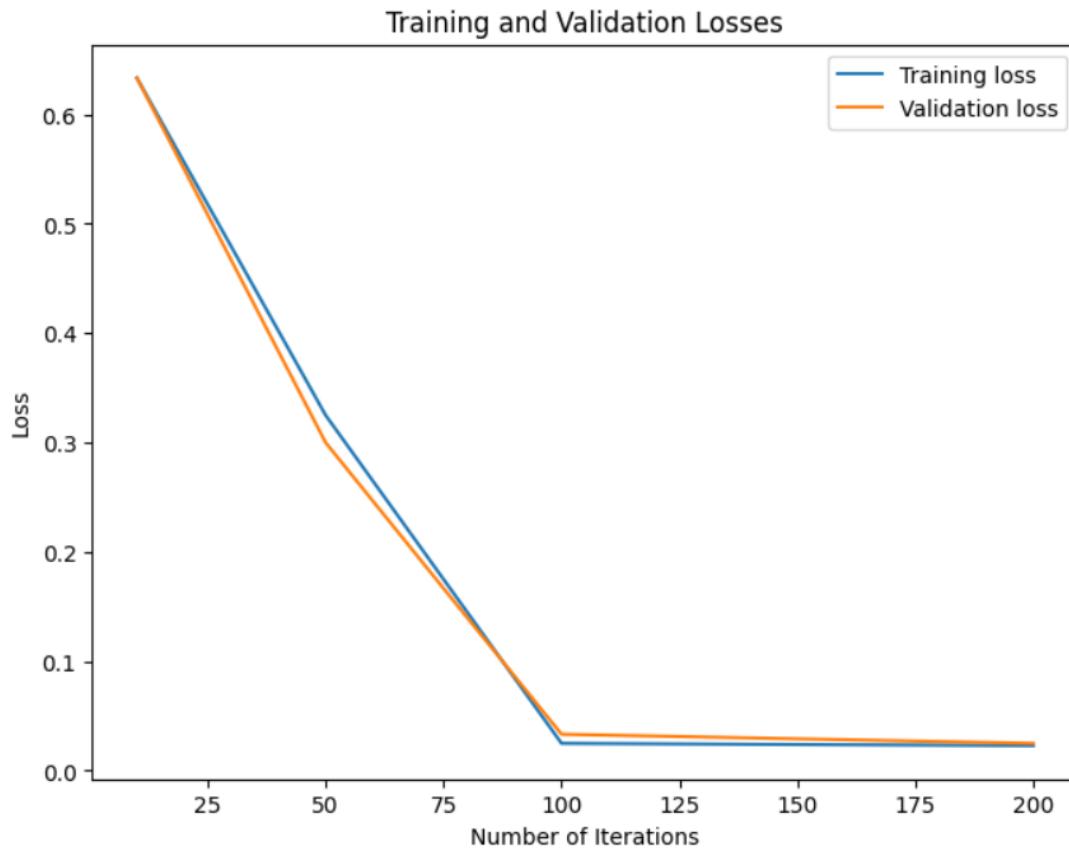
# Calculate the accuracy on the test set
test_accuracy = model.score(x_test, y_test)

# Plotting the training and validation losses
plt.figure(figsize=(8, 6))
plt.plot([10, 50, 100, 200], train_losses, label='Training loss')
plt.plot([10, 50, 100, 200], valid_losses, label='Validation loss')
plt.xlabel('Number of Iterations')
plt.ylabel('Loss')
plt.title('Training and Validation Losses')
plt.legend()
plt.show()

print(f"Test Accuracy: {test_accuracy}")

```

مشاهده خروجی:



حال مدل را بدون استفاده از کتابخانه‌های آماده پایتون تعریف می‌کنیم:

ابتدا کلاس MyMLPClassifier را تعریف می‌کنیم:

```
class MyMLPClassifier:  
    def __init__(self, input_size, hidden_layer_size, output_size,  
learning_rate=0.001, max_iter=200):  
        self.input_size = input_size  
        self.hidden_layer_size = hidden_layer_size  
        self.output_size = output_size  
        self.learning_rate = learning_rate  
        self.max_iter = max_iter  
  
        self.W1 = np.random.rand(self.input_size, self.hidden_layer_size)  
        self.b1 = np.random.rand(1, self.hidden_layer_size)  
        self.W2 = np.random.rand(self.hidden_layer_size, self.output_size)  
        self.b2 = np.random.rand(1, self.output_size)  
  
    def relu(self, x):  
        return np.maximum(0, x)  
  
    def softmax(self, x):  
        exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))  
        return exp_x / exp_x.sum(axis=1, keepdims=True)  
  
    def fit(self, X, y):  
        for _ in range(self.max_iter):  
            # Forward propagation  
            z1 = np.dot(X, self.W1) + self.b1  
            a1 = self.relu(z1)  
            z2 = np.dot(a1, self.W2) + self.b2  
            y_pred = self.softmax(z2)  
  
            # Backpropagation  
            delta3 = y_pred  
            delta3[range(len(X)), y] -= 1  
            dW2 = np.dot(a1.T, delta3)  
            db2 = np.sum(delta3, axis=0, keepdims=True)  
            delta2 = np.dot(delta3, self.W2.T) * (a1 > 0)  
            dW1 = np.dot(X.T, delta2)  
            db1 = np.sum(delta2, axis=0)  
  
            # Update weights  
            self.W1 -= self.learning_rate * dW1  
            self.b1 -= self.learning_rate * db1
```

```

        self.W2 -= self.learning_rate * dW2
        self.b2 -= self.learning_rate * db2

    def predict(self, X):
        z1 = np.dot(X, self.W1) + self.b1
        a1 = self.relu(z1)
        z2 = np.dot(a1, self.W2) + self.b2
        y_pred = self.softmax(z2)
        return np.argmax(y_pred, axis=1)

    def score(self, X, y):
        y_pred = self.predict(X)
        return np.mean(y_pred == y)

```

گفتیم که این کلاس MyMLPClassifier یک طبقه‌بندی‌کننده MLP ایجاد می‌کند. اجزای اصلی آن عبارتند از:

\_\_init\_\_: برای تعریف پارامترهای مدل مانند اندازه ورودی، لایه مخفی و خروجی.

Fit: الگوریتم آموزش با پیش‌برد، پس‌برد و بهروزرسانی وزن‌ها.

Predict: پیش‌بینی برای داده‌های جدید.

Score: محاسبه دقت بر روی داده‌های آزمون.

کارکرد اصلی آن عبارت است از:

۱. تعریف شبکه با وزن‌های تصادفی

۲. پیش‌برد داده‌ها در شبکه برای پیش‌بینی

۳. محاسبه خطأ و بهروزرسانی وزن‌ها

۴. تکرار گام ۲ و ۳ تا حد اکثر تکرار

۵. استفاده از مدل آموزش‌دیده برای پیش‌بینی

در ادامه مدلمان را تعریف می‌کنیم:

```

model = MyMLPClassifier(input_size=4, hidden_layer_size=100,
output_size=3, learning_rate=0.001, max_iter=200)
model.fit(x_train, y_train)

```

```
model.score(x_test, y_test)
```

مشاهده دقت:

```
1.0
```

سپس با استفاده از کدی که در قسمت قبل (تعريف مدل لجستیک رگرسیون بدون استفاده از کتابخانه‌های آماده پایتون) نوشتیم، ماتریس درهم‌ریختگی را تعریف می‌کنیم:

```
# Making predictions on the test set
y_pred_2 = model.predict(x_test)

# Calculating confusion matrix
cf_matrix = np.zeros((2,2))

y_test_1 = np.round(y_test).astype(int)
y_pred_2 = np.round(y_pred_2).astype(int)

classes = np.unique(y_test_1)
n_classes = len(classes)

cf_matrix = np.zeros((n_classes, n_classes))

for i in range(len(y_test)):
    cf_matrix[y_test_1[i], y_pred_2[i]] += 1

# Plotting confusion matrix as a heatmap with fitted text
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues',
annot_kws={"size": 12})
sns.heatmap(cf_matrix, annot=True, fmt='.2f', cmap='Blues',
annot_kws={"size": 12})

# Get the axis to modify layout
plt.gca().set_yticks(len(np.unique(y_test_1)), 0)

plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
```

```

# Save the plot as PNG
plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()

# Printing classification report
print("Classification Report:")

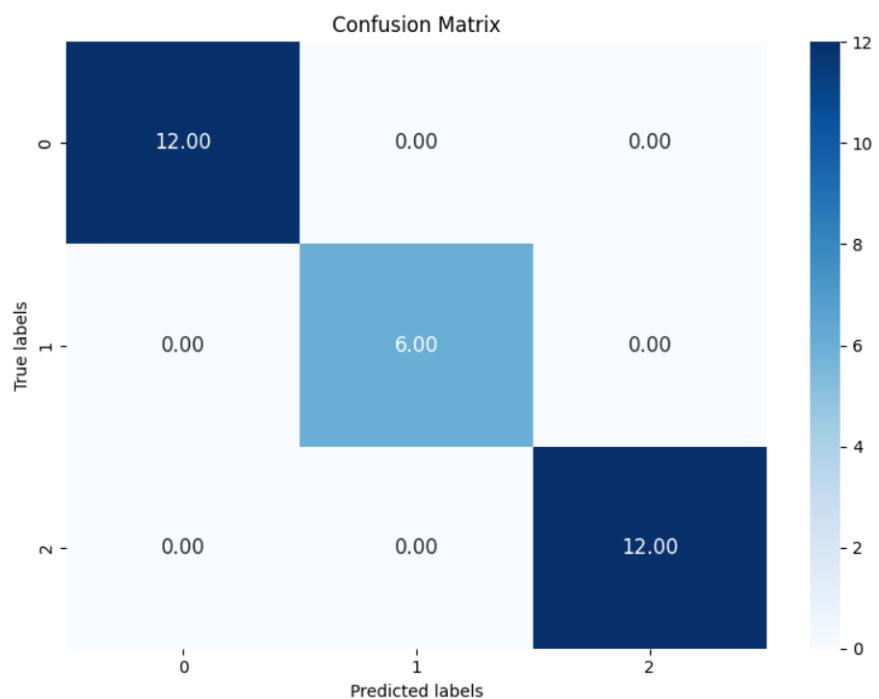
tp = 0
fp = 0
fn = 0
tn = 0

for i in range(len(y_test)):
    if y_test[i]==1 and y_pred_2[i]==1:
        tp += 1
    elif y_test[i]==1 and y_pred_2[i]==0:
        fn += 1
    elif y_test[i]==0 and y_pred_2[i]==1:
        fp += 1
    elif y_test[i]==0 and y_pred_2[i]==0:
        tn += 1

print("Precision: ", tp/(tp+fp))
print("Recall: ", tp/(tp+fn))
print("Accuracy: ", (tp+tn) / (tp+fp+fn+tn))

```

مشاهده ماتریس درهم ریختگی :



بررسی شاخص‌ها و ویژگی‌های مدل:

```
Classification Report:  
Precision: 1.0  
Recall: 1.0  
Accuracy: 1.0
```

با توجه به گزارش دیده شده در خروجی می‌توان گفت که دقت و صحت ما ۱ بوده و حتی نسبت به حالت قبل (مدلی که با استفاده از کتابخانه‌های آماده پایتون طراحی کردیم) نیز بهتر است. پس این مدل مناسب‌تر می‌باشد.

### مدل بر اساس RBF:

ابتدا با استفاده از کتابخانه‌های آماده پایتون مدلمان را بر اساس شبکه‌های عصبی پایه شعاعی تعریف می‌کنیم برای اینکار از الگوریتم ماشین بردار پشتیبان (SVM) که یکی از الگوریتم‌های یادگیری ماشین برای مسائل طبقه‌بندی و رگرسیون می‌باشد، استفاده می‌کنیم:

```
from sklearn.svm import SVC  
  
# Assuming you've trained your model already  
model = SVC(kernel='rbf', random_state=12)  
model.fit(x_train, y_train)
```

در این قسمت از کد از کتابخانه scikit-learn در پایتون برای پیاده‌سازی SVM استفاده شده است. ابتدا ما از کلاس SVC از sklearn.svm برای ایجاد یک مدل SVM استفاده کرده‌ایم. مقدار kernel='rbf' برای مشخص کردن اینکه از kernel رابط (Radial Basis Function) برای SVM استفاده شود و مقدار random\_state=12 برای تولید اعداد تصادفی استفاده شده است.

سپس با استفاده از متدهای fit، مدل را روی داده‌های آموزش (x\_train و y\_train) آموزش داده‌ایم.

در ادامه با روشی که در مدل‌های قبلی نیز استفاده کردیم، ماتریس درهم‌ریختگی و اطلاعات مدل را فراخوانی می‌کنیم:

```
# Making predictions on the test set  
y_pred = model.predict(x_test)  
  
# Calculating confusion matrix  
cf_matrix = confusion_matrix(y_test, y_pred)
```

```

# Plotting confusion matrix as a heatmap with fitted text
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues',
annot_kws={"size": 12})

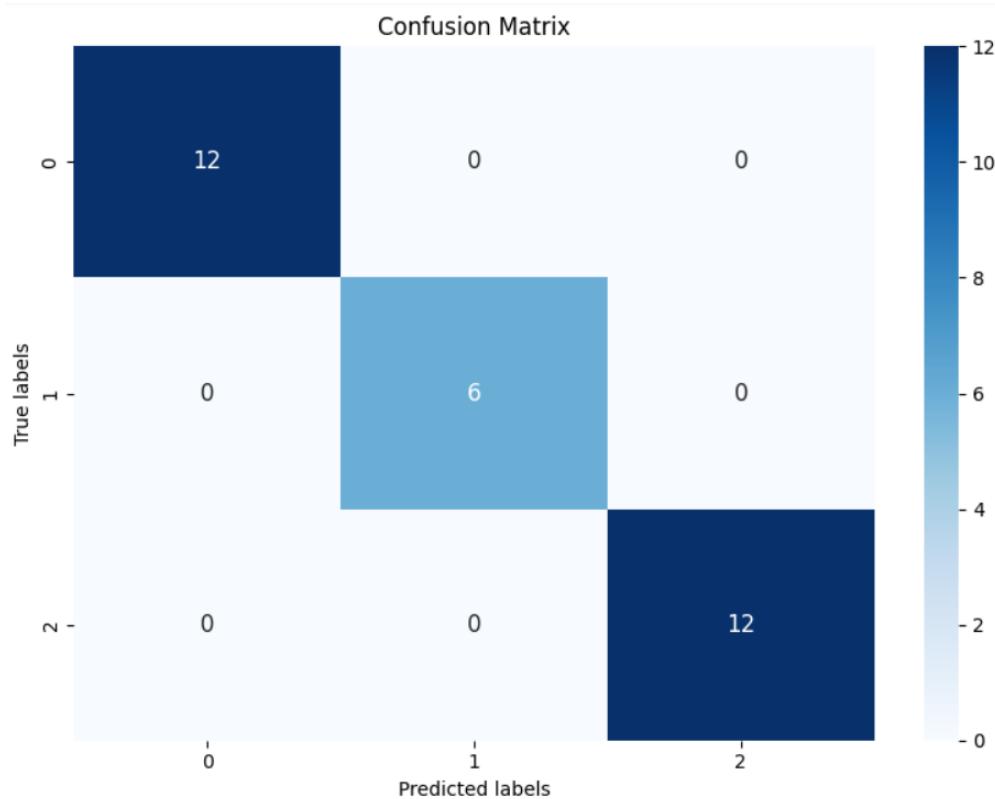
# Get the axis to modify layout
plt.gca().set_ylim(len(np.unique(y_test)), 0) # Fix for matplotlib 3.1.1
and 3.1.2
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Save the plot as PNG
plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()

# Printing classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

```

مشاهده ماتریس درهم ریختگی:



بررسی شاخص‌ها و ویژگی‌های مدل:

Classification Report:					
	precision	recall	f1-score	support	
0	1.00	1.00	1.00	12	
1	1.00	1.00	1.00	6	
2	1.00	1.00	1.00	12	
accuracy			1.00	30	
macro avg	1.00	1.00	1.00	30	
weighted avg	1.00	1.00	1.00	30	

با توجه به گزارش داده شده می‌توان نکات زیر را تحلیل کرد:

precision: برای طبقه ۰ و ۱ که ۱ است نشان می‌دهد تمام آیتم‌های طبقه بندی شده برای این طبقات صحیح بوده است.

recall: این ویژگی برای تمام طبقات ۱ است که نشان‌دهنده بازیابی بالای مدل است.

f1-score: مقار آن ۱ است که نشان‌دهنده عملکرد عالی مدل است.

Accuracy: نشان میدهد تقریباً ۱۰۰ درصد از آیتم‌ها را مدل به درستی طبقه بندی کرده است.

در کل می‌توان گفت مدل ما برای آموزش داده‌هایمان مناسب می‌باشد.

البته این سوال را می‌توان با استفاده از روش گاووسی گسسته (Gaussian Process) نیز حل کرد:

```
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF

# Assuming you've trained your model already
kernel = 0.1 * RBF(0.1) # Define the RBF kernel
model = GaussianProcessClassifier(kernel=kernel, random_state=12)
model.fit(x_train, y_train)
```

ابتدا یک kernel از نوع RBF (Radial Basis Function) با پارامترهای مشخص شده ایجاد می‌کنیم. سپس از این kernel برای تعریف مدل GaussianProcessClassifier استفاده می‌کنیم. مقدار random\_state=12 نیز برای تعیین seed برای تولید اعداد تصادفی استفاده شده است.

این مدل برای مسائلی که داده‌ها به صورت غیرخطی توزیع شده‌اند و رویش‌های پیچیده‌ای دارند، به خصوص مواردی که توزیع داده‌ها را نمی‌توان با یک مدل خطی توصیف کرد، مناسب است.

در ادامه ماتریس درهم‌ریختگی را نیز تعریف می‌کنیم:

```
# Making predictions on the test set
y_pred = model.predict(x_test)

# Calculating confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)

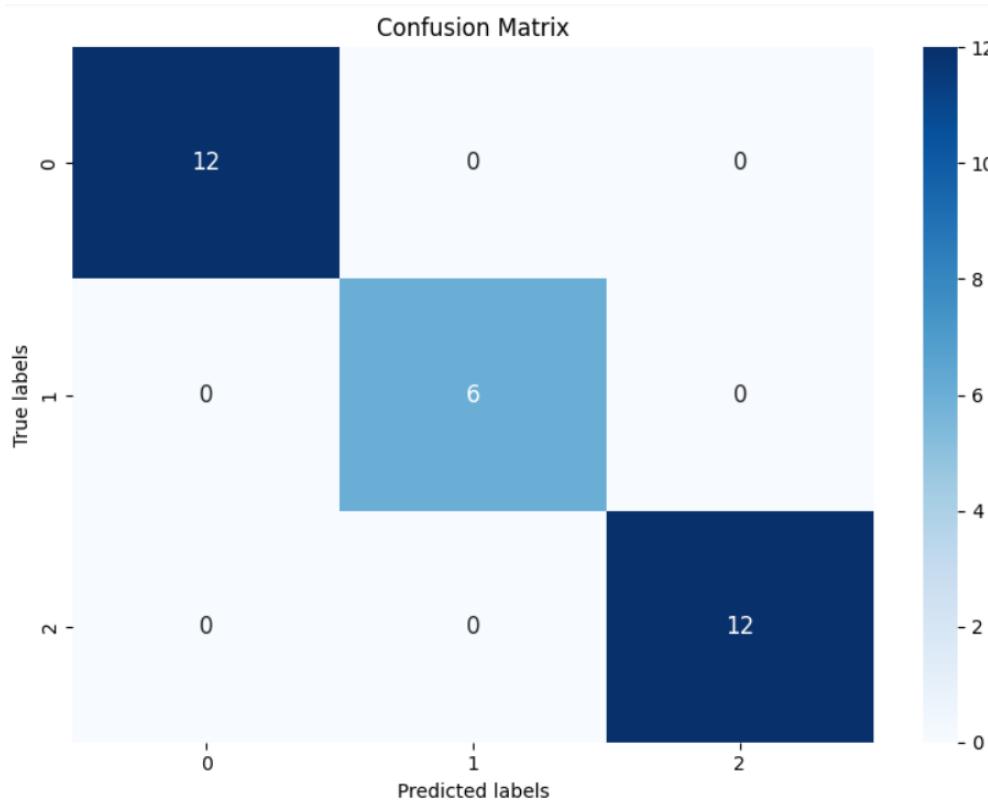
# Plotting confusion matrix as a heatmap with fitted text
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues',
annot_kws={"size": 12})

# Get the axis to modify layout
plt.gca().set_yticks(len(np.unique(y_test)), 0) # Fix for matplotlib 3.1.1
and 3.1.2
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Save the plot as PNG
plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()

# Printing classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

مشاهده ماتریس درهم‌ریختگی:



بررسی شاخص‌ها و ویژگی‌های مدل:

Classification Report:					
	precision	recall	f1-score	support	
0	1.00	1.00	1.00	12	
1	1.00	1.00	1.00	6	
2	1.00	1.00	1.00	12	
accuracy			1.00	30	
macro avg	1.00	1.00	1.00	30	
weighted avg	1.00	1.00	1.00	30	

با توجه به گزارش داده شده می‌توان نکات زیر را تحلیل کرد:

precision: برای طبقه ۰ و ۱ که ۱ است نشان می‌دهد تمام آیتم‌های طبقه بندی شده برای این طبقات صحیح بوده است.

recall: این ویژگی برای تمام طبقات ۱ است که نشان‌دهنده بازیابی بالای مدل است.

f1-score: مقار آن ۱ است که نشان‌دهنده عملکرد عالی مدل است.

: نشان میدهد تقریباً ۱۰۰ درصد از آیتم ها را مدل به درستی طبقه بندی کرده است. Accuracy

این مدل نیز همانند مدل قبلی RBF برای آموزش داده هایمان مناسب می باشد.

همانند قسمت لجستیک رگرسیون و RBF اینجا نیز می توان میزان همبستگی هر طبقه را بر اساس درصد نشان

داد:

```
# Assuming you've trained your model already
model = SVC(kernel='rbf', random_state=12)
model.fit(x_train, y_train)

# Making predictions on the test set
y_pred = model.predict(x_test)

# Calculating confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)

# Calculating percentages for each cell
cf_matrix_percent = cf_matrix.astype('float') / cf_matrix.sum(axis=1)[:, np.newaxis] * 100

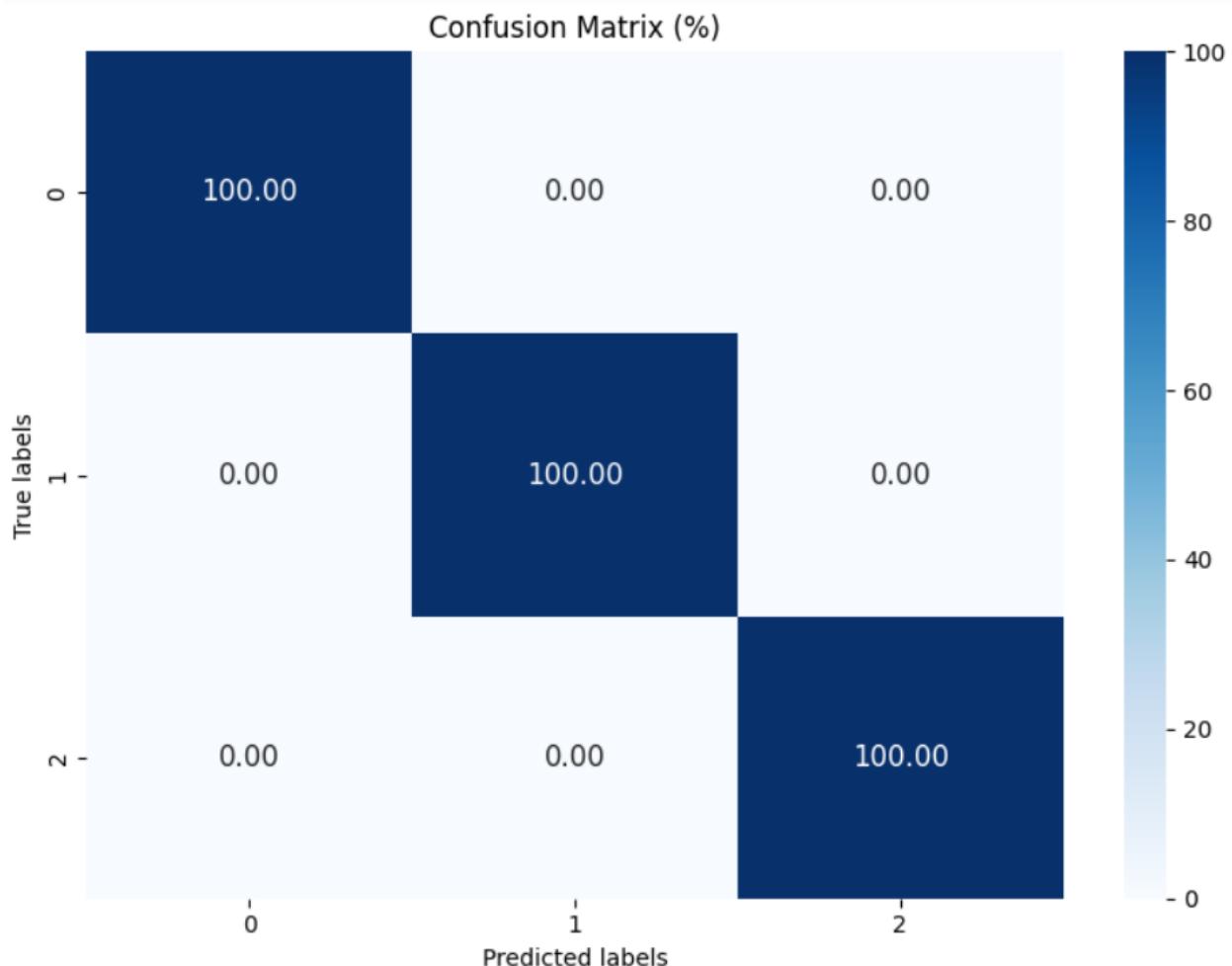
# Plotting confusion matrix as a heatmap with percentages
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix_percent, annot=True, fmt='.2f', cmap='Blues',
            annot_kws={"size": 12})

# Get the axis to modify layout
plt.gca().set_yticks(len(np.unique(y_test)), 0) # Fix for matplotlib 3.1.1
and 3.1.2
plt.title('Confusion Matrix (%)')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Save the plot as PNG
plt.tight_layout()
plt.savefig('confusion_matrix_percentage.png', dpi=300)
plt.show()

# Printing classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

مشاهده خروجی:



حال به بررسی این قسمت بدون استفاده از کتابخانه‌های آماده پایتون و سایکیت لرن می‌پردازیم:

در قسمت زیر، ۲ کلاس برای پیاده‌سازی یک مدل رگرسیون گاوی (Gaussian Process Regression) ایجاد شده است. ابتدا کلاس‌های زیر را تعریف می‌کنیم:

```
import numpy as np

class RBF:
    def __init__(self, sigma):
        self.sigma = sigma

    def calc(self, x1, x2):
        return np.exp(-np.sum((x1-x2)**2) / (2*self.sigma**2))

class GPR:
```

```

def __init__(self, kernel):
    self.kernel = kernel

def fit(self, X, y):
    self.X = X
    self.y = y

def predict(self, X_test):
    predictions = np.zeros((len(X_test), 1))

    for i, x_test in enumerate(X_test):
        for j, x_train in enumerate(self.X):
            k = self.kernel.calc(x_test, x_train)
            predictions[i] += k * self.y[j]

    predictions[i] = np.sign(predictions[i])

    return predictions

def score(self, X_test, y_test):
    y_pred = self.predict(X_test)
    accuracy = np.mean(y_pred == y_test)
    return accuracy

```

کلاس RBF برای تعریف یک kernel از نوع RBF استفاده شده است. این کلاس دارای یک متدهای calc است که برای محاسبه مقدار kernel بین دو نقطه استفاده می‌شود.

کلاس GPR برای تعریف مدل رگرسیون گاوی استفاده شده است. این کلاس دارای متدهای fit برای آموزش مدل، predict برای انجام پیش‌بینی‌ها بر روی داده‌های تست، و score برای محاسبه دقیق مدل بر روی داده‌های تست است.

در متدهای predict، برای هر نقطه از داده‌های تست، مقدار پیش‌بینی با استفاده از kernel محاسبه می‌شود و سپس مقدار نهایی با توجه به مقادیر y مربوط به داده‌های آموزش محاسبه می‌شود.

در متدهای score نیز، دقیق مدل بر روی داده‌های تست محاسبه شده و برگردانده می‌شود.

حال می‌توانیم مدلمان را تعریف کنیم:

```
kernel = RBF(0.1)
```

```
model = GPR(kernel)
model.fit(x_train, y_train)
model.score(x_test,y_test)
```

در کد بالا، ابتدا یک kernel از نوع RBF با پارامتر  $\sigma$ =0.1 ایجاد شده و سپس یک مدل GPR با استفاده از این kernel ایجاد شده است. سپس مدل بر روی داده‌های آموزش (x\_train) و (y\_train) آموزش داده شده و در نهایت دقت مدل بر روی داده‌های تست با استفاده از متدهای score محاسبه شده و برگردانده می‌شود.

مشاهده دقت:

```
0.2
```

مشاهده می‌شود که دقت ما پایین است و احتمالاً مدل ما نتواند عمل طبقه‌بندی را به خوبی انجام دهد.

حال ماتریس درهم‌ریختگی را با برگرفتن از روش‌هایی که در رگرسیون لجستیک و MLP بدون استفاده از کتابخانه‌های آماده پایتون ارائه دادیم، می‌یابیم:

```
# Making predictions on the test set
y_pred_3 = model.predict(x_test)

# Calculating confusion matrix
cf_matrix = np.zeros((2,2))

y_test_1 = np.round(y_test).astype(int)
y_pred_3 = np.round(y_pred_3).astype(int)

classes = np.unique(y_test_1)
n_classes = len(classes)

cf_matrix = np.zeros((n_classes, n_classes))

for i in range(len(y_test)):
    cf_matrix[y_test_1[i], y_pred_3[i]] += 1

# Plotting confusion matrix as a heatmap with fitted text
plt.figure(figsize=(8, 6))
#sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues',
#annot_kws={"size": 12})
sns.heatmap(cf_matrix, annot=True, fmt='%.2f', cmap='Blues',
annot_kws={"size": 12})
```

```

# Get the axis to modify layout
plt.gca().set_ylimits(len(np.unique(y_test_1)), 0)

plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Save the plot as PNG
plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()

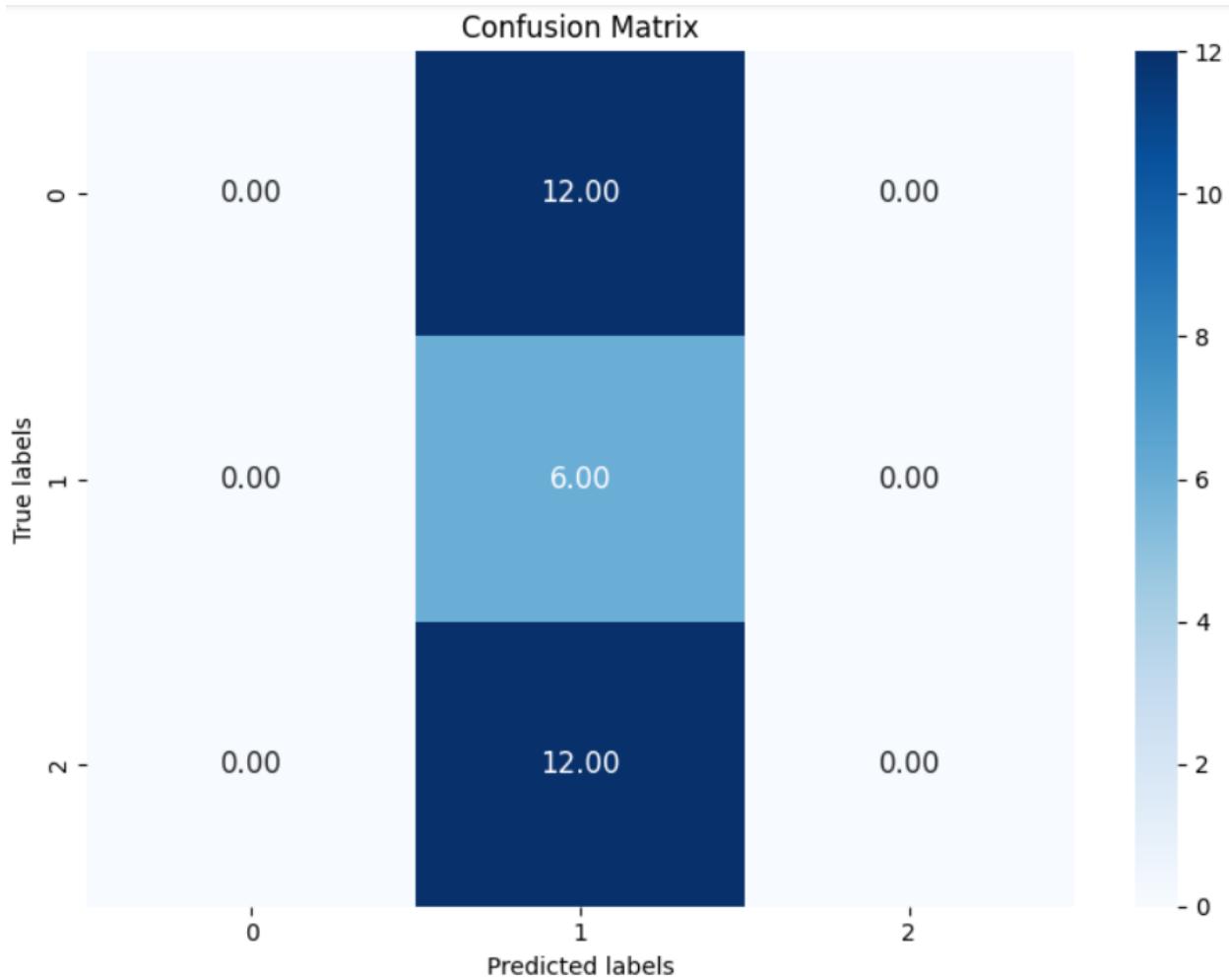
# Printing classification report
print("Classification Report:")

tp = 0
fp = 0
fn = 0
tn = 0
for i in range(len(y_test)):
    if y_test[i]==1 and y_pred_3[i]==1:
        tp += 1
    elif y_test[i]==1 and y_pred_3[i]==0:
        fn += 1
    elif y_test[i]==0 and y_pred_3[i]==1:
        fp += 1
    elif y_test[i]==0 and y_pred_3[i]==0:
        tn += 1

print("Precision: ", tp/(tp+fp))
print("Recall: ", tp/(tp+fn))
print("Accuracy: ", (tp+tn) / (tp+fp+fn+tn))

```

مشاهده ماتریس درهم ریختگی :



بررسی شاخص‌ها و ویژگی‌های مدل:

**Classification Report:**  
 Precision: 0.333333333333333  
 Recall: 1.0  
 Accuracy: 0.333333333333333

با توجه به گزارش دیده شده در خروجی می‌توان گفت که دقت و صحت ما ۳۳٪ بوده و نسبت به حالت قبل (مدلی که با استفاده از کتابخانه‌های آماده پایتون طراحی کردیم) پایین‌تر است. پس در روش RBF برای تعریف مدل، استفاده از کتابخانه‌های آماده پایتون مناسب‌تر می‌باشد.